Appendix C32
Invalidity of U.S. Patent 8,880,862 based on Jones

15. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system.	Jones, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 1 (Preamble) and 1.1 above.	

16. The method of claim 14, wherein the operating system comprises: a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 14, wherein the operating system comprises: a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

17. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.	Jones, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 15 above.	

Appendix C32
Invalidity of U.S. Patent 8,880,862 based on Jones

19.1 The method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:	Jones, as evidenced by the example citations below, discloses "the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

19.2 associating the accessed boot data that is not associated with the boot data list to the boot data list.	Jones, as evidenced by the example citations below, discloses "associating the accessed boot data that is not associated with the boot data list to the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, associating the accessed boot data that is not associated with the boot data list to the boot data list.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 2 above.	

23. The method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 16 above.	

24. The method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.	Jones, as evidenced by the example citations below, discloses “the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Jones discloses this limitation: *See* Claim 15 above.	

27. The method of claim 1, wherein the memory comprises: a physical memory.	Jones, as evidenced by the example citations below, discloses "the method of claim 1, wherein the memory comprises: a physical memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

28. The method of claim 1, wherein the operating system comprises: a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 1, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 16 above.	

29. The method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program.	Jones, as evidenced by the example citations below, discloses "the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15 and 17 above.	

31. The method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access.	Jones, as evidenced by the example citations below, discloses "the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

Appendix C32
Invalidity of U.S. Patent 8,880,862 based on Jones

32. The method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form.	Jones, as evidenced by the example citations below, discloses “the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

33. The method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form.	Jones, as evidenced by the example citations below, discloses "the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 32 above.	

35. The method of claim 5, wherein the compressed boot data represents a plurality of files.	Jones, as evidenced by the example citations below, discloses “the method of claim 5, wherein the compressed boot data represents a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

36. The method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system.	Jones, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15 and 17 above.	

39. The method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory.	Jones, as evidenced by the example citations below, discloses “the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

40. The method of claim 36, wherein the operating system comprises: a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 36, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 36, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

43. The method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access.	Jones, as evidenced by the example citations below, discloses "the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 31 above.	

44. The method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Jones, as evidenced by the example citations below, discloses "the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 32 above.	

45. The method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Jones, as evidenced by the example citations below, discloses "the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 32 and 33 above.	

47. The system of claim 6, wherein the boot data in the compressed form represents a plurality of files.	Jones, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

48. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system.	Jones, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

49. The system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form.	Jones, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 10 above.	

50. The system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form.	Jones, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: "Custom video and audio hardware for the set-top box contains a MPEG-2 decoder, NTSC (the U.S. and Japanese analog television encoding standard) encoders & decoders, a tuner, and an audio mixer." Jones, §3. "Everything from server machines and ATM switches to real-time MPEG-2 encoders and OS software was precisely duplicated." Jones, §11.	

51. The system of claim 6, wherein the first memory comprises: a physical memory.	Jones, as evidenced by the example citations below, discloses "the system of claim 6, wherein the first memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the first memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 27 and 39 above.	

52. The system of claim 6, wherein the boot data the compressed form comprises: a plurality of files.	Jones, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

53. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program.	Jones, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

59. The method of claim 8, wherein the operating system in the compressed form represents a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

60. The method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system.	Jones, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

63. The method of claim 8, wherein the second memory comprises: a physical memory.	Jones, as evidenced by the example citations below, discloses "the method of claim 8, wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 27 and 38 above.	

64. The method of claim 8, wherein the operating system comprises: a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

65. The method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program.	Jones, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

67. The method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access.	Jones, as evidenced by the example citations below, discloses "the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 31 above.	

Appendix C32
Invalidity of U.S. Patent 8,880,862 based on Jones

71. The method of claim 11, wherein the boot data in the compressed form represents a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

72. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

75. The method of claim 11, wherein the memory comprises: a physical memory.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 27 above.	

76. The method of claim 11, wherein the operating system comprises: a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

77. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

79. The method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 31 above.	

80. The method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 32, 33 and 49 above.	

81. The method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form.	Jones, as evidenced by the example citations below, discloses "the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 32, 33 and 49 above.	

83. The method of claim 13, wherein the boot data in the compressed form represents a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

84. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Jones, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

87. The method of claim 13, wherein the memory comprises: a physical memory.	Jones, as evidenced by the example citations below, discloses “the method of claim 13, wherein the memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Jones discloses this limitation: *See* Claim 27 above.	

88. The method of claim 13, wherein the operating system comprises: a plurality of files.	Jones, as evidenced by the example citations below, discloses "the method of claim 13, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 16 and 23 above.	

89. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program.	Jones, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 15, 17 and 24 above.	

91. The method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access.	Jones, as evidenced by the example citations below, discloses "the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 31 above.	

92. The method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Jones, as evidenced by the example citations below, discloses "the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 32, 33 and 49 above.	

93. The method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Jones, as evidenced by the example citations below, discloses "the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claim 32, 33, and 49 above.	

97.1 The method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list,	Jones, as evidenced by the example citations below, discloses "the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 9.1-9.3 and 19 above.	

97.2 and wherein the updating comprises: associating the additional compressed boot data with the boot data list.	Jones, as evidenced by the example citations below, discloses "and wherein the updating comprises: associating the additional compressed boot data with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, and wherein the updating comprises: associating the additional compressed boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 1.4.1, 2, 4, 5.6, and 8.6 above.	

98. The method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list.	Jones, as evidenced by the example citations below, discloses "the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

107. The method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.	Jones, as evidenced by the example citations below, discloses “the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Jones discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

108. The method of claim 2, further comprising: compressing at least a portion of the additional boot data.	Jones, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: compressing at least a portion of the additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: compressing at least a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 1.1, 5, 9.1 and 8 above.	

Appendix C32
Invalidity of U.S. Patent 8,880,862 based on Jones

109. The method of claim 108, further comprising: storing the compressed additional boot data.	Jones, as evidenced by the example citations below, discloses "the method of claim 108, further comprising: storing the compressed additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 108, further comprising: storing the compressed additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Jones discloses this limitation: *See* Claims 5.1, 8.1 and 9.1 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

The publication Magstar and IBM 3590 High Performance Tape Subsystem Technical Guide, November 1996 ("Magstar") invalidates claims 1-6, 8-11, 13-17, 19, 23-24, 27-29, 31-33, 35-36, 39-40, 43-45, 47-53, 59-60, 63-65, 67, 71-72, 75-77, 79-81, 83-84, 87-89, 91-93, 97-98, and 107-109 of United States Patent No. 8,880,862 ("the '862 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '862 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

1 (Preamble) A method for providing accelerated loading of an operating system in a computer system, the method comprising:	Magstar, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, the method comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system in a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: "The drive data rate at 9 MB/s is three times faster than the IBM 3480, 3490, or 3490E tape drives. This increase in drive data rate, together with the built-in data compression capability, makes it possible to utilize more effectively the full capability of a 20 MB/s fast-and-wide SCSI or a 17 MB/s ESCON channel." Magstar at 25. "If the compression ratio achieved is greater than 3:1, the attainable throughput is constrained by the channel speed. Nevertheless, for most applications using conventional DASD, the data rate attained is determined not by the tape drive speed but by another system or application component." Magstar, 133. "Two data patterns were used in this study to provide compaction ratios of approximately 1.5:1 and 3:1. A third set of results for the uncompacted data case was achieved by setting the JCL parameter TRTCH=NOCOMP to turn off the IDRC compression. Table 8 on page 134, taken from WSC Flash 9108 IBM 3490E Tape Subsystem Performance, which documents the study, relates to a subsystem such as the subsystem shown in the diagram, using an ESCON channel, an 8MB buffer, and with five drives active." Magstar, 133.	

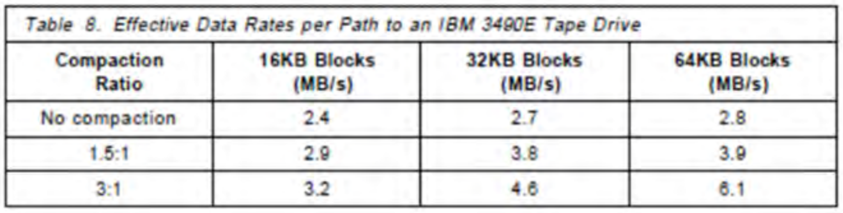
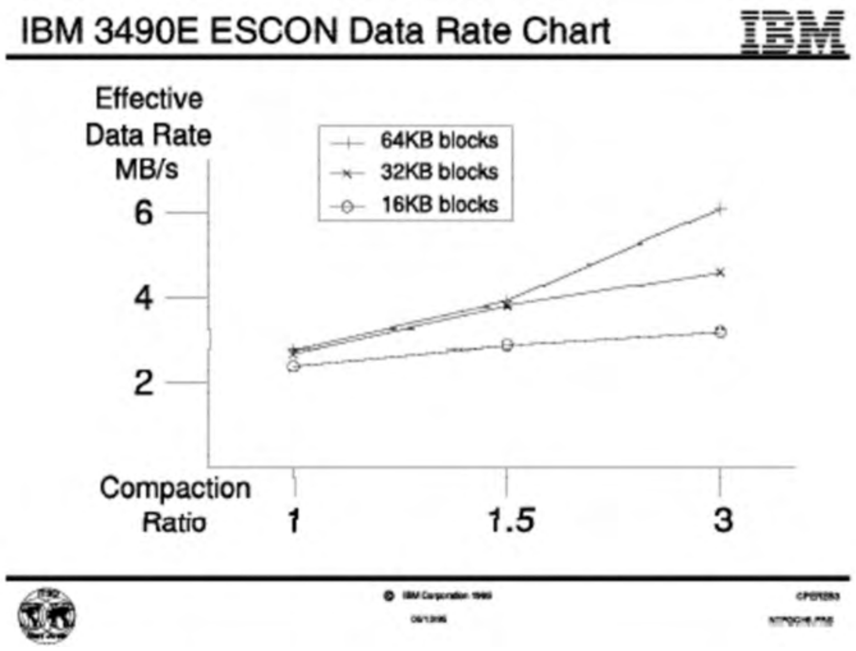


Table 8. Effective Data Rates per Path to an IBM 3490E Tape Drive

Compaction Ratio	16KB Blocks (MB/s)	32KB Blocks (MB/s)	64KB Blocks (MB/s)
No compaction	2.4	2.7	2.8
1.5:1	2.9	3.8	3.9
3:1	3.2	4.6	6.1

Magstar, Table 8.



Magstar, Figure 60.

> "The measured data rates illustrate two points about effective data rates as opposed to native data rates in tape subsystems:
>
> - When the data is not compressed, the effective data rate is limited by the drive speed, that is, approximately 3MB/s. However, when the data is compressed, the maximum data rate is approximately 6MB/s.
> - The data rate is dependent on block size. With a block size of 16KB and a compression ratio of 3:1, the maximum data rate is 3.2MB/s."

Magstar, 135. *See also generally*, Magstar 135-145.

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2

m/s). A buffer is used and the data compressed before it is written to.”   Magstar, 12.

1.1 loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;	Magstar, as evidenced by the example citations below, discloses "loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's[3] class of encoding methods."

Magstar, 25. *See also generally* Magstar, 25-26 at "Improved compression."

> "It is during the process of transferring data from the host channel to the buffer that the data can be compressed using the BAC algorithm component of IDRC."

Magstar, 132.

> "The compression achieved using IDRC is made up of two factors:
>
> - Automatic reblocking of the data to a block size of 128KB, which has a more marked effect on small block sizes
> - Applying the BAC algorithm to the data, the effectiveness of which depends on the randomness of the data, and whether or not it has already been compressed (for example, by hardware or software data compression in the host)."

Magstar, 132. *See also* generally Magstar, 132 at § 6.1.2 and §6.9.

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2 m/s). A buffer is used and the data compressed before it is written to."

Magstar, 12.

1.2 accessing the loaded portion of the boot data in the compressed form from memory;	Magstar, as evidenced by the example citations below, discloses "accessing the loaded portion of the boot data in the compressed form from memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the boot data in the compressed form from memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's[3] class of encoding methods."

Magstar, 25. *See also generally* Magstar, 25-26 at "Improved compression."

> "It is during the process of transferring data from the host channel to the buffer that the data can be compressed using the BAC algorithm component of IDRC."

Magstar, 132.

> "The compression achieved using IDRC is made up of two factors:
>
> - Automatic reblocking of the data to a block size of 128KB, which has a more marked effect on small block sizes
> - Applying the BAC algorithm to the data, the effectiveness of which depends on the randomness of the data, and whether or not it has already been compressed (for example, by hardware or software data compression in the host)."

Magstar, 132. *See also generally* Magstar, 132 at § 6.1.2 and §6.9.

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2 m/s). A buffer is used and the data compressed before it is written to."

Magstar, 12.

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

1.3 decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and	Magstar, as evidenced by the example citations below, discloses "decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

> "The drive data rate at 9 MB/s is three times faster than the IBM 3480, 3490, or 3490E tape drives. This increase in drive data rate, together with the built-in data compression capability, makes it possible to utilize more effectively the full capability of a 20 MB/s fast-and-wide SCSI or a 17 MB/s ESCON channel."

Magstar at 25.

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's[3] class of encoding methods."

Magstar, 25. *See also generally* Magstar, 25-26 at "Improved compression."

> "If the compression ratio achieved is greater than 3:1, the attainable throughput is constrained by the channel speed. Nevertheless, for most applications using conventional DASD, the data rate attained is determined not by the tape drive speed but by another system or application component."

Magstar, 133.

> "Two data patterns were used in this study to provide compaction ratios of approximately 1.5:1 and 3:1. A third set of results for the uncompacted data case was achieved by setting the JCL parameter TRTCH=NOCOMP to turn off the IDRC compression. Table 8 on page 134, taken from WSC Flash 9108 IBM 3490E Tape Subsystem Performance, which documents the study, relates to a

> subsystem such as the subsystem shown in the diagram, using an ESCON channel, an 8MB buffer, and with five drives active."

Magstar, 133.

> "Two data patterns were used in this study to provide compaction ratios of approximately 1.5:1 and 3:1. A third set of results for the uncompacted data case was achieved by setting the JCL parameter TRTCH=NOCOMP to turn off the IDRC compression. Table 8 on page 134, taken from WSC Flash 9108 IBM 3490E Tape Subsystem Performance, which documents the study, relates to a subsystem such as the subsystem shown in the diagram, using an ESCON channel, an 8MB buffer, and with five drives active."

Magstar, 133.

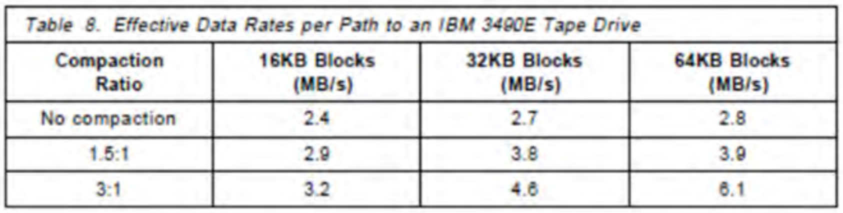
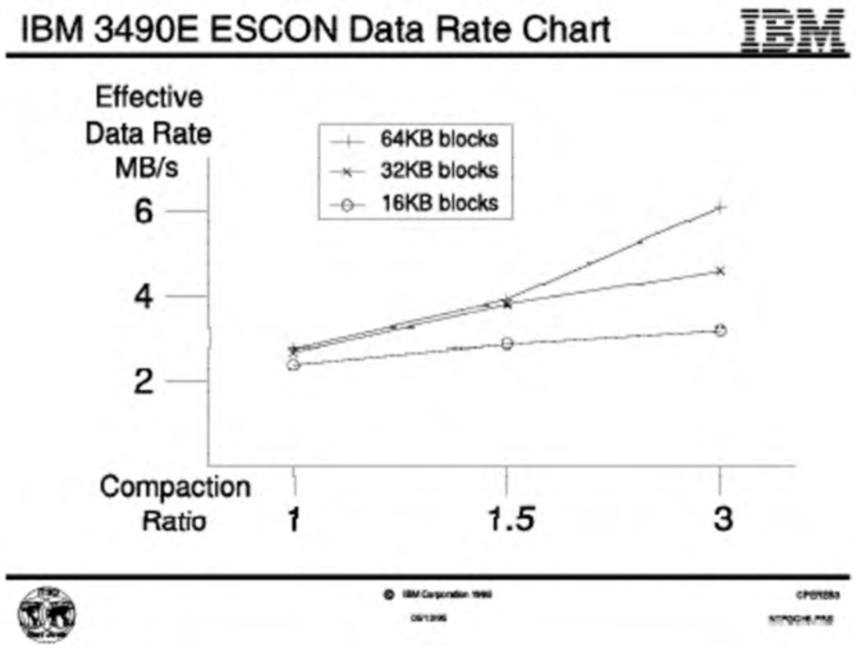


Table 8. Effective Data Rates per Path to an IBM 3490E Tape Drive

Compaction Ratio	16KB Blocks (MB/s)	32KB Blocks (MB/s)	64KB Blocks (MB/s)
No compaction	2.4	2.7	2.8
1.5:1	2.9	3.8	3.9
3:1	3.2	4.6	6.1

Magstar, Table 8.



Magstar, Figure 60.

> "The measured data rates illustrate two points about effective data rates as opposed to native data rates in tape subsystems:
>
> - When the data is not compressed, the effective data rate is limited by the drive speed, that is, approximately 3MB/s. However, when the data is compressed, the maximum data rate is approximately 6MB/s.
> - The data rate is dependent on block size. With a block size of 16KB and a compression ratio of 3:1, the maximum data rate is 3.2MB/s."
> -

Magstar, 135. *See also generally*, Magstar 135-145

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2 m/s). A buffer is used and the data compressed before it is written to."

Magstar, 12.

1.4.1 updating the boot data list,	Magstar, as evidenced by the example citations below, discloses “updating the boot data list,”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

1.4.2 wherein the decompressed portion of boot data comprises a portion of the operating system.	Magstar, as evidenced by the example citations below, discloses "wherein the decompressed portion of boot data comprises a portion of the operating system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the decompressed portion of boot data comprises a portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

2. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list.	Magstar, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

3. The method of claim 1, wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list.	Magstar, as evidenced by the example citations below, discloses "wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein removing an association of additional boot data that is associated with the boot data list from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

4. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data.	Magstar, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list; and compressing a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.4.1, and 2 above.	

5 (Preamble) A method for booting a computer system, the method comprising:	Magstar, as evidenced by the example citations below, discloses "a method for booting a computer system, the method comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1-1.4.2 above.	

5.1 storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory;	Magstar, as evidenced by the example citations below, discloses "storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

5.2 loading the stored compressed boot data from the first memory;	Magstar, as evidenced by the example citations below, discloses "loading the stored compressed boot data from the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the stored compressed boot data from the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

5.3 accessing the loaded compressed boot data;	Magstar, as evidenced by the example citations below, discloses "accessing the loaded compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.2 above.	

5.4 decompressing the accessed compressed boot data;	Magstar, as evidenced by the example citations below, discloses "decompressing the accessed compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.3 above.	

5.5 utilizing the decompressed boot data to at least partially boot the computer system;	Magstar, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to at least partially boot the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

See Claims 1.3 and 1.4.2 above.

See also

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's[3] class of encoding methods."

Magstar, 25. *See also generally* Magstar, 25-26 at "Improved compression."

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2 m/s). A buffer is used and the data compressed before it is written to."

Magstar, 12.

5.6 updating the boot data list;	Magstar, as evidenced by the example citations below, discloses “updating the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

5.7 wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.	Magstar, as evidenced by the example citations below, discloses "wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1-1.3 above.	

6 (Preamble) A system comprising: a processor;	Magstar, as evidenced by the example citations below, discloses "a system comprising: a processor:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system comprising: a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

6.1 a memory; and	Magstar, as evidenced by the example citations below, discloses "a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.2 a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor	Magstar, as evidenced by the example citations below, discloses "a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.1 and 1.2 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

6.3 wherein the processor is configured:	Magstar, as evidenced by the example citations below, discloses "wherein the processor is configured:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the processor is configured), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

6.4 to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory,	Magstar, as evidenced by the example citations below, discloses “to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

6.5 to access the loaded portion of the boot data in the compressed form,	Magstar, as evidenced by the example citations below, discloses "to access the loaded portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to access the loaded portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.2 above.	

6.6 to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data, and	Magstar, as evidenced by the example citations below, discloses "to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.3 above.	

6.7 to update the boot data list.	Magstar, as evidenced by the example citations below, discloses “to update the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to update the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

8 (Preamble) A method of loading an operating system for booting a computer system, comprising:	Magstar, as evidenced by the example citations below, discloses "A method of loading an operating system for booting a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method of loading an operating system for booting a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1 (Preamble) above.	

8.1 storing a portion of the operating system in a compressed form in a first memory;	Magstar, as evidenced by the example citations below, discloses "storing a portion of the operating system in a compressed form in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing a portion of the operating system in a compressed form in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 6.2 above.	

8.2 loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list;	Magstar, as evidenced by the example citations below, discloses "loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

8.3 accessing the loaded portion of the operating system from the second memory in the compressed form;	Magstar, as evidenced by the example citations below, discloses "accessing the loaded portion of the operating system from the second memory in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the operating system from the second memory in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.2 above.	

8.4 decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system;	Magstar, as evidenced by the example citations below, discloses "decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.5 utilizing the decompressed portion of the operating system to at least partially boot the computer system; and	Magstar, as evidenced by the example citations below, discloses "utilizing the decompressed portion of the operating system to at least partially boot the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed portion of the operating system to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

See Claims 1.3 and 1.4.2 above.

See also

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's[3] class of encoding methods."

Magstar, 25. *See also generally* Magstar, 25-26 at "Improved compression."

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2 m/s). A buffer is used and the data compressed before it is written to."

Magstar, 12.

8.6 updating the boot data list,	Magstar, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

8.7 wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form.	Magstar, as evidenced by the example citations below, discloses "wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1-1.3 and 5.7 above.	

9.1 The method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list; and	Magstar, as evidenced by the example citations below, discloses "the method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, compressing an additional portion of the operating system that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

9.2 storing the additional portion of the operating system in the first memory, and	Magstar, as evidenced by the example citations below, discloses “storing the additional portion of the operating system in the first memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing the additional portion of the operating system in the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 8.1 above.	

9.3 wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system.	Magstar, as evidenced by the example citations below, discloses "wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 8.5 above.	

10. The method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder.	Magstar, as evidenced by the example citations below, discloses "the method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

See Claim 9.1 above.

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's[3] class of encoding methods."

Magstar, 25. *See also generally* Magstar, 25-26 at "Improved compression."

> "It is during the process of transferring data from the host channel to the buffer that the data can be compressed using the BAC algorithm component of IDRC."

Magstar, 132.

> "The compression achieved using IDRC is made up of two factors:
>
> - Automatic reblocking of the data to a block size of 128KB, which has a more marked effect on small block sizes
> - Applying the BAC algorithm to the data, the effectiveness of which depends on the randomness of the data, and whether or not it has already been compressed (for example, by hardware or software data compression in the host)."

Magstar, 132. *See also generally* Magstar, 132 at § 6.1.2 and §6.9.

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2

> m/s). A buffer is used and the data compressed before it is written to."

Magstar, 12.

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's3 class of encoding methods."

Magstar, 25.

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

11 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Magstar, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1 (Preamble) above.	

11.1 loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system;	Magstar, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

11.2 accessing the loaded boot data in compressed form from the memory;	Magstar, as evidenced by the example citations below, discloses “accessing the loaded boot data in compressed form from the memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in compressed form from the memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.2 above.	

11.3 decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Magstar, as evidenced by the example citations below, discloses "decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.3 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

11.4 utilizing the decompressed boot data to load at least a portion of the operating system for the computer system; and	Magstar, as evidenced by the example citations below, discloses “utilizing the decompressed boot data to load at least a portion of the operating system for the computer system”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to load at least a portion of the operating system for the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Magstar discloses this limitation:

> “The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson’s[3] class of encoding methods.”

Magstar, 25. *See also generally* Magstar, 25-26 at “Improved compression.”

> “New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2 m/s). A buffer is used and the data compressed before it is written to.”

Magstar, 12.

11.5 updating the boot data list.	Magstar, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

13 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Magstar, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1 (Preamble) above.	

13.1 loading boot data in a compressed form that is associated with a boot data list from a boot device;	Magstar, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

13.2 accessing the loaded boot data in the compressed form;	Magstar, as evidenced by the example citations below, discloses "accessing the loaded boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.2 above.	

13.3 decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Magstar, as evidenced by the example citations below, discloses "decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.3 above.	

13.4 updating the boot data list.	Magstar, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

14 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Magstar, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1 (Preamble) above.	

14.1 accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list;	Magstar, as evidenced by the example citations below, discloses "accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.1 and 1.2 above.	

14.2 loading the boot data into a memory; and	Magstar, as evidenced by the example citations below, discloses "loading the boot data into a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the boot data into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.1 above.	

14.3 servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form; and	Magstar, as evidenced by the example citations below, discloses "servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

See Claims 1.2 and 1.3 above.

See also

> "The drive data rate at 9 MB/s is three times faster than the IBM 3480, 3490, or 3490E tape drives. This increase in drive data rate, together with the built-in data compression capability, makes it possible to utilize more effectively the full capability of a 20 MB/s fast-and-wide SCSI or a 17 MB/s ESCON channel."

Magstar at 25.

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's[3] class of encoding methods."

Magstar, 25. *See also generally* Magstar, 25-26 at "Improved compression."

> "If the compression ratio achieved is greater than 3:1, the attainable throughput is constrained by the channel speed. Nevertheless, for most applications using conventional DASD, the data rate attained is determined not by the tape drive speed but by another system or

application component."

Magstar, 133.

> "Two data patterns were used in this study to provide compaction ratios of approximately 1.5:1 and 3:1. A third set of results for the uncompacted data case was achieved by setting the JCL parameter TRTCH=NOCOMP to turn off the IDRC compression. Table 8 on page 134, taken from WSC Flash 9108 IBM 3490E Tape Subsystem Performance, which documents the study, relates to a subsystem such as the subsystem shown in the diagram, using an ESCON channel, an 8MB buffer, and with five drives active."

Magstar, 133.

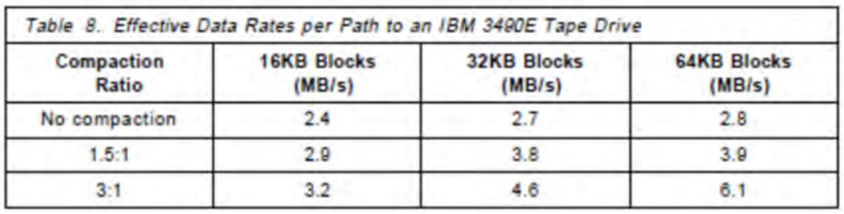
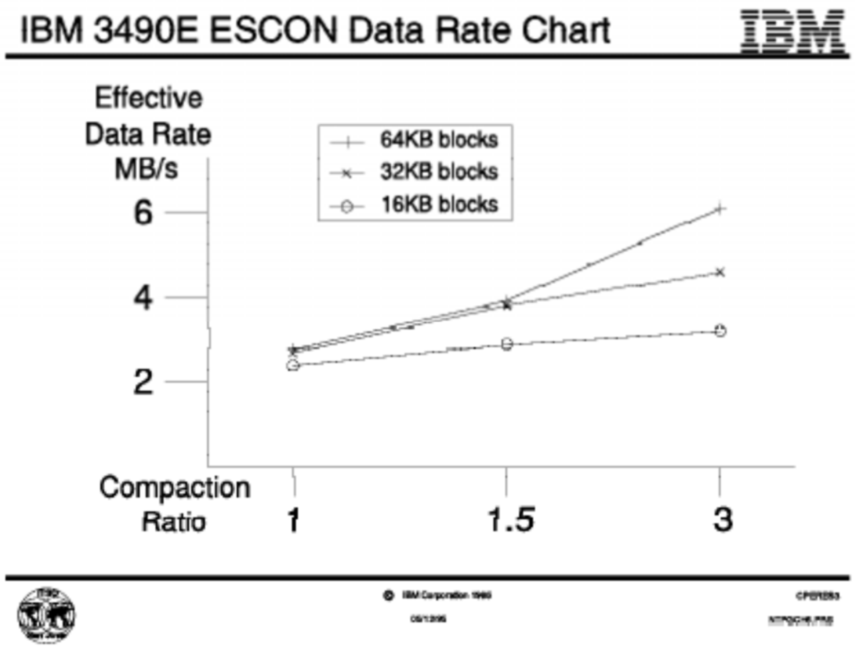


Table 8. Effective Data Rates per Path to an IBM 3490E Tape Drive

Compaction Ratio	16KB Blocks (MB/s)	32KB Blocks (MB/s)	64KB Blocks (MB/s)
No compaction	2.4	2.7	2.8
1.5:1	2.9	3.8	3.9
3:1	3.2	4.6	6.1

Magstar, Table 8.



Magstar, Figure 60.

> "The measured data rates illustrate two points about effective data rates as opposed to native data rates in tape subsystems:
>
> - When the data is not compressed, the effective data rate is limited by the drive speed, that is, approximately 3MB/s. However, when the data is compressed, the maximum data rate is approximately 6MB/s.
> - The data rate is dependent on block size. With a block size of 16KB and a compression ratio of 3:1, the maximum data rate is 3.2MB/s."

Magstar, 135. *See also generally*, Magstar 135-145

> "The IBM Magstar Virtual Tape Server Controller, Tape Volume Cache, and the IBM 3590 Magstar tape drives, together with the required housing, make up the IBM Magstar Virtual Tape Server (VTS) subsystem, allowing automatic utilization of the Magstar cartridge storage capacity and the drive data rate of 9 MB/s."

Magstar, 150. *See also generally* Magstar, 150-211.

> "New longitudinal technology now announced by IBM (16-track Serpentine Interleaved Longitudinal Recording) significantly improves tape performance and transfer rates without changing the tape speed (2 m/s). A buffer is used and the data compressed before it is written to."

Magstar, 12.

14.4 updating the boot data list.	Magstar, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 1.4.1 above.	

15. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system.	Magstar, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1 (Preamble) and 1.1 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

16. The method of claim 14, wherein the operating system comprises: a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 14, wherein the operating system comprises: a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

17. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.	Magstar, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 15 above.	

19.1 The method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:	Magstar, as evidenced by the example citations below, discloses "the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

19.2 associating the accessed boot data that is not associated with the boot data list to the boot data list.	Magstar, as evidenced by the example citations below, discloses "associating the accessed boot data that is not associated with the boot data list to the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, associating the accessed boot data that is not associated with the boot data list to the boot data list.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 2 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

23. The method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 16 above.	

24. The method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.	Magstar, as evidenced by the example citations below, discloses “the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 15 above.	

27. The method of claim 1, wherein the memory comprises: a physical memory.	Magstar, as evidenced by the example citations below, discloses "the method of claim 1, wherein the memory comprises: a physical memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See generally*, Magstar, 2-269.	

28. The method of claim 1, wherein the operating system comprises: a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 1, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 16 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

29. The method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program.	Magstar, as evidenced by the example citations below, discloses "the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15 and 17 above.	

31. The method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access.	Magstar, as evidenced by the example citations below, discloses "the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

32. The method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form.	Magstar, as evidenced by the example citations below, discloses "the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Magstar discloses this limitation:

> "The Magstar tape drive also has an improved compression algorithm (Ziv-Lempel) which is called IBMLZ1 and will be more efficient than the binary arithmetic compression (BAC) algorithm used in the IBM 3480 and 3490 Tape Subsystem□s IDRC."

Magstar, 7.

> "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson□s3 class of encoding methods. For the IBM 3590 IBMLZ1 compression, a 1024 bytes of history buffer is used. This implementation differs from the IDRC used in the IBM 3490E, which is based on BAC. The IBMLZ1 algorithm is expected to be more effective than IDRC."

Magstar, 25.

> "The IBMLZ1 compression algorithm is designed for robust and highly efficient compression. Key design objectives for the IBMLZ1 algorithm are:
>
> - Hardware execution efficiency: the hardware architecture should use as few machine cycles as possible to compress or decompress a byte. The architecture should maintain low complexity and use silicon technology effectively. In addition, the smallest number of machine cycles for each byte (this number is called CPB) should be used to compress or decompress data.
>
> - Robust compression: achieve good coding efficiency for broad

applications. - Minimum system integration overhead: the maximum benefit from compression is achieved when the compression can be performed without performance loss. So the algorithm is capable of running at channel speed (20 MB/s for SCSI)." Magstar 25-26.

33. The method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form.	Magstar, as evidenced by the example citations below, discloses "the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 32 above.	

35. The method of claim 5, wherein the compressed boot data represents a plurality of files.	Magstar, as evidenced by the example citations below, discloses “the method of claim 5, wherein the compressed boot data represents a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

36. The method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system.	Magstar, as evidenced by the example citations below, discloses “the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15 and 17 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

39. The method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory.	Magstar, as evidenced by the example citations below, discloses “the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

40. The method of claim 36, wherein the operating system comprises: a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 36, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 36, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

43. The method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access.	Magstar, as evidenced by the example citations below, discloses “the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 31 above.	

44. The method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Magstar, as evidenced by the example citations below, discloses “the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 32 above.	

45. The method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Magstar, as evidenced by the example citations below, discloses "the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 32 and 33 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

47. The system of claim 6, wherein the boot data in the compressed form represents a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

48. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system.	Magstar, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

49. The system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form.	Magstar, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 10 above.	

50. The system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form.	Magstar, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: "The IBMLZ1 compression algorithm used in the IBM 3590 is based on the Ziv-Lempel algorithm2 and Jackson's3 class of encoding methods." Magstar, 25.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

51. The system of claim 6, wherein the first memory comprises: a physical memory.	Magstar, as evidenced by the example citations below, discloses "the system of claim 6, wherein the first memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the first memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 27 and 39 above.	

52. The system of claim 6, wherein the boot data the compressed form comprises: a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

53. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program.	Magstar, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

59. The method of claim 8, wherein the operating system in the compressed form represents a plurality of files.	Magstar, as evidenced by the example citations below, discloses “the method of claim 8, wherein the operating system in the compressed form represents a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

60. The method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system.	Magstar, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

63. The method of claim 8, wherein the second memory comprises: a physical memory.	Magstar, as evidenced by the example citations below, discloses "the method of claim 8, wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 27 and 38 above.	

64. The method of claim 8, wherein the operating system comprises: a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

65. The method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program.	Magstar, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

67. The method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access.	Magstar, as evidenced by the example citations below, discloses "the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 31 above.	

71. The method of claim 11, wherein the boot data in the compressed form represents a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

72. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Magstar, as evidenced by the example citations below, discloses “the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

75. The method of claim 11, wherein the memory comprises: a physical memory.	Magstar, as evidenced by the example citations below, discloses “the method of claim 11, wherein the memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 27 above.	

76. The method of claim 11, wherein the operating system comprises: a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 11, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

77. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program.	Magstar, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

79. The method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access.	Magstar, as evidenced by the example citations below, discloses "the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 31 above.	

80. The method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form.	Magstar, as evidenced by the example citations below, discloses “the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claims 32, 33 and 49 above.	

81. The method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form.	Magstar, as evidenced by the example citations below, discloses "the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 32, 33 and 49 above.	

83. The method of claim 13, wherein the boot data in the compressed form represents a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

84. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Magstar, as evidenced by the example citations below, discloses “the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

87. The method of claim 13, wherein the memory comprises: a physical memory.	Magstar, as evidenced by the example citations below, discloses "the method of claim 13, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 27 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

88. The method of claim 13, wherein the operating system comprises: a plurality of files.	Magstar, as evidenced by the example citations below, discloses "the method of claim 13, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 16 and 23 above.	

89. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program.	Magstar, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 15, 17 and 24 above.	

91. The method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access.	Magstar, as evidenced by the example citations below, discloses “the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claim 31 above.	

92. The method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Magstar, as evidenced by the example citations below, discloses "the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 32, 33 and 49 above.	

93. The method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Magstar, as evidenced by the example citations below, discloses "the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claim 32, 33, and 49 above.	

Appendix C33
Invalidity of U.S. Patent 8,880,862 based on Magstar

97.1 The method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list,	Magstar, as evidenced by the example citations below, discloses "the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 9.1-9.3 and 19 above.	

97.2 and wherein the updating comprises: associating the additional compressed boot data with the boot data list.	Magstar, as evidenced by the example citations below, discloses "and wherein the updating comprises: associating the additional compressed boot data with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, and wherein the updating comprises: associating the additional compressed boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.4.1, 2, 4, 5.6, and 8.6 above.	

98. The method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list.	Magstar, as evidenced by the example citations below, discloses "the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

107. The method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.	Magstar, as evidenced by the example citations below, discloses “the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

108. The method of claim 2, further comprising: compressing at least a portion of the additional boot data.	Magstar, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: compressing at least a portion of the additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: compressing at least a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 1.1, 5, 9.1 and 8 above.	

109. The method of claim 108, further comprising: storing the compressed additional boot data.	Magstar, as evidenced by the example citations below, discloses "the method of claim 108, further comprising: storing the compressed additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 108, further comprising: storing the compressed additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Magstar discloses this limitation: *See* Claims 5.1, 8.1 and 9.1 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

The publication Mealey, B, IBM, An IP.com Prior Art Database Technical Disclosure, January, 1992 ("Mealey") invalidates claims 1-6, 8-11, 13-17, 19, 23-24, 27-29, 31-33, 35-36, 39-40, 43-45, 47-53, 59-60, 63-65, 67, 71-72, 75-77, 79-81, 83-84, 87-89, 91-93, 97-98, and 107-109 of United States Patent No. 8,880,862 ("the '862 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '862 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

1 (Preamble) A method for providing accelerated loading of an operating system in a computer system, the method comprising:	Mealey, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, the method comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system in a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image. The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded. The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it. Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete." Mealey, 1.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

1.1 loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;	Mealey, as evidenced by the example citations below, discloses "loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

1.2 accessing the loaded portion of the boot data in the compressed form from memory;	Mealey, as evidenced by the example citations below, discloses "accessing the loaded portion of the boot data in the compressed form from memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the boot data in the compressed form from memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

1.3 decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and	Mealey, as evidenced by the example citations below, discloses "decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

1.4.1 updating the boot data list,	Mealey, as evidenced by the example citations below, discloses "updating the boot data list,"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

1.4.2 wherein the decompressed portion of boot data comprises a portion of the operating system.	Mealey, as evidenced by the example citations below, discloses "wherein the decompressed portion of boot data comprises a portion of the operating system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the decompressed portion of boot data comprises a portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

2. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list.	Mealey, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

3. The method of claim 1, wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list.	Mealey, as evidenced by the example citations below, discloses "wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein removing an association of additional boot data that is associated with the boot data list from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

4. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data.	Mealey, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list; and compressing a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.4.1, and 2 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

5 (Preamble) A method for booting a computer system, the method comprising:	Mealey, as evidenced by the example citations below, discloses "a method for booting a computer system, the method comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1-1.4.2 above.	

5.1 storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory;	Mealey, as evidenced by the example citations below, discloses "storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

5.2 loading the stored compressed boot data from the first memory;	Mealey, as evidenced by the example citations below, discloses "loading the stored compressed boot data from the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the stored compressed boot data from the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

5.3 accessing the loaded compressed boot data;	Mealey, as evidenced by the example citations below, discloses "accessing the loaded compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.2 above.	

5.4 decompressing the accessed compressed boot data;	Mealey, as evidenced by the example citations below, discloses "decompressing the accessed compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.3 above.	

5.5 utilizing the decompressed boot data to at least partially boot the computer system;	Mealey, as evidenced by the example citations below, discloses “utilizing the decompressed boot data to at least partially boot the computer system”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Mealey discloses this limitation:

See Claims 1.3 and 1.4.2 above.

5.6 updating the boot data list;	Mealey, as evidenced by the example citations below, discloses “updating the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

5.7 wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.	Mealey, as evidenced by the example citations below, discloses "wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1-1.3 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

6 (Preamble) A system comprising: a processor;	Mealey, as evidenced by the example citations below, discloses “a system comprising: a processor:”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system comprising: a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: “Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image. The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded. The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it. Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete.” Mealey, 1.	

6.1 a memory; and	Mealey, as evidenced by the example citations below, discloses "a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.2 a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor	Mealey, as evidenced by the example citations below, discloses "a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.3 wherein the processor is configured:	Mealey, as evidenced by the example citations below, discloses “wherein the processor is configured:”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the processor is configured), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Mealey discloses this limitation:

> “Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete.”

Mealey, 1.

6.4 to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory,	Mealey, as evidenced by the example citations below, discloses "to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

6.5 to access the loaded portion of the boot data in the compressed form,	Mealey, as evidenced by the example citations below, discloses "to access the loaded portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to access the loaded portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.2 above.	

6.6 to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data, and	Mealey, as evidenced by the example citations below, discloses "to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.3 above.	

6.7 to update the boot data list.	Mealey, as evidenced by the example citations below, discloses "to update the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to update the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

8 (Preamble) A method of loading an operating system for booting a computer system, comprising:	Mealey, as evidenced by the example citations below, discloses “A method of loading an operating system for booting a computer system, comprising:”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method of loading an operating system for booting a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1 (Preamble) above.	

8.1 storing a portion of the operating system in a compressed form in a first memory;	Mealey, as evidenced by the example citations below, discloses "storing a portion of the operating system in a compressed form in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing a portion of the operating system in a compressed form in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 6.2 above.	

8.2 loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list;	Mealey, as evidenced by the example citations below, discloses “loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

8.3 accessing the loaded portion of the operating system from the second memory in the compressed form;	Mealey, as evidenced by the example citations below, discloses "accessing the loaded portion of the operating system from the second memory in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the operating system from the second memory in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.2 above.	

8.4 decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system;	Mealey, as evidenced by the example citations below, discloses "decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.5 utilizing the decompressed portion of the operating system to at least partially boot the computer system; and	Mealey, as evidenced by the example citations below, discloses "utilizing the decompressed portion of the operating system to at least partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed portion of the operating system to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.6 updating the boot data list,	Mealey, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

8.7 wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form.	Mealey, as evidenced by the example citations below, discloses “wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1-1.3 and 5.7 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

9.1 The method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list; and	Mealey, as evidenced by the example citations below, discloses "the method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, compressing an additional portion of the operating system that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

9.2 storing the additional portion of the operating system in the first memory, and	Mealey, as evidenced by the example citations below, discloses “storing the additional portion of the operating system in the first memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing the additional portion of the operating system in the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claim 8.1 above.	

9.3 wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system.	Mealey, as evidenced by the example citations below, discloses "wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 8.5 above.	

10. The method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder.	Mealey, as evidenced by the example citations below, discloses "the method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 9.1 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

11 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Mealey, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1 (Preamble) above.	

11.1 loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system;	Mealey, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

11.2 accessing the loaded boot data in compressed form from the memory;	Mealey, as evidenced by the example citations below, discloses "accessing the loaded boot data in compressed form from the memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in compressed form from the memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.2 above.	

11.3 decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Mealey, as evidenced by the example citations below, discloses "decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.3 above.	

11.4 utilizing the decompressed boot data to load at least a portion of the operating system for the computer system; and	Mealey, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to load at least a portion of the operating system for the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to load at least a portion of the operating system for the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image. The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded. The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it. Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete." Mealey, 1.	

11.5 updating the boot data list.	Mealey, as evidenced by the example citations below, discloses “updating the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

13 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Mealey, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1 (Preamble) above.	

13.1 loading boot data in a compressed form that is associated with a boot data list from a boot device;	Mealey, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.1 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

13.2 accessing the loaded boot data in the compressed form;	Mealey, as evidenced by the example citations below, discloses "accessing the loaded boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.2 above.	

13.3 decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Mealey, as evidenced by the example citations below, discloses "decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.3 above.	

13.4 updating the boot data list.	Mealey, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

14 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Mealey, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1 (Preamble) above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

14.1 accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list;	Mealey, as evidenced by the example citations below, discloses "accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1 and 1.2 above.	

14.2 loading the boot data into a memory; and	Mealey, as evidenced by the example citations below, discloses “loading the boot data into a memory”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the boot data into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Mealey discloses this limitation:

See Claim 1.1 above.

14.3 servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form; and	Mealey, as evidenced by the example citations below, discloses "servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.2 and 1.3 above.	

14.4 updating the boot data list.	Mealey, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 1.4.1 above.	

15. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system.	Mealey, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1 (Preamble) and 1.1 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

16. The method of claim 14, wherein the operating system comprises: a plurality of files.	Mealey, as evidenced by the example citations below, discloses “the method of claim 14, wherein the operating system comprises: a plurality of files.”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: “Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image. The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded. The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it. Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete.” Mealey, 1.	

17. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.	Mealey, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 15 above.	

19.1 The method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:	Mealey, as evidenced by the example citations below, discloses "the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

19.2 associating the accessed boot data that is not associated with the boot data list to the boot data list.	Mealey, as evidenced by the example citations below, discloses "associating the accessed boot data that is not associated with the boot data list to the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, associating the accessed boot data that is not associated with the boot data list to the boot data list.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 2 above.	

23. The method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 16 above.	

24. The method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.	Mealey, as evidenced by the example citations below, discloses “the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claim 15 above.	

27. The method of claim 1, wherein the memory comprises: a physical memory.	Mealey, as evidenced by the example citations below, discloses "the method of claim 1, wherein the memory comprises: a physical memory."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

28. The method of claim 1, wherein the operating system comprises: a plurality of files.	Mealey, as evidenced by the example citations below, discloses “the method of claim 1, wherein the operating system comprises: a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claim 16 above.	

29. The method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program.	Mealey, as evidenced by the example citations below, discloses "the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15 and 17 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

31. The method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access.	Mealey, as evidenced by the example citations below, discloses "the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

32. The method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form.	Mealey, as evidenced by the example citations below, discloses "the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image. The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded. The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it. Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete." Mealey, 1.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

33. The method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form.	Mealey, as evidenced by the example citations below, discloses "the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 32 above.	

35. The method of claim 5, wherein the compressed boot data represents a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

36. The method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system.	Mealey, as evidenced by the example citations below, discloses “the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15 and 17 above.	

39. The method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory.	Mealey, as evidenced by the example citations below, discloses "the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Mealey discloses this limitation:

> "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image.
> The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded.
> The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it.
> Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete."

Mealey, 1.

40. The method of claim 36, wherein the operating system comprises: a plurality of files.	Mealey, as evidenced by the example citations below, discloses “the method of claim 36, wherein the operating system comprises: a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 36, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

43. The method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access.	Mealey, as evidenced by the example citations below, discloses "the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 31 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

44. The method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Mealey, as evidenced by the example citations below, discloses "the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 32 above.	

45. The method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Mealey, as evidenced by the example citations below, discloses "the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 32 and 33 above.	

47. The system of claim 6, wherein the boot data in the compressed form represents a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

48. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system.	Mealey, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

49. The system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form.	Mealey, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 10 above.	

50. The system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form.	Mealey, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: "Disclosed is an approach that reduces resources and time required for system initialization. This is accomplished by applying a data compression algorithm to the text and data of a boot-image. The boot-image is divided into two portions: text and data. The text is compressed and bound with a decompression program. It is decompressed immediately after being loaded. The data portion is a RAM disk file system. The file system is compressed at the block level. It is accessed by a pseudo device driver that compresses on writes and decompresses on reads. The file system remains compressed during system initialization reducing the amount of memory required to hold it. Applying compression to a boot image can produce a significant time savings when booting from a slow device, such as a busy LAN or floppy disk. It also reduces the size of the media required to hold the image. Finally, it can reduce the amount of memory required to hold the boot image until system initialization is complete." Mealey, 1.	

51. The system of claim 6, wherein the first memory comprises: a physical memory.	Mealey, as evidenced by the example citations below, discloses “the system of claim 6, wherein the first memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the first memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Mealey discloses this limitation: *See* Claims 27 and 39 above.	

52. The system of claim 6, wherein the boot data the compressed form comprises: a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

53. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program.	Mealey, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

59. The method of claim 8, wherein the operating system in the compressed form represents a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

60. The method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system.	Mealey, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

63. The method of claim 8, wherein the second memory comprises: a physical memory.	Mealey, as evidenced by the example citations below, discloses "the method of claim 8, wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 27 and 38 above.	

64. The method of claim 8, wherein the operating system comprises: a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

65. The method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program.	Mealey, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

67. The method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access.	Mealey, as evidenced by the example citations below, discloses "the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 31 above.	

71. The method of claim 11, wherein the boot data in the compressed form represents a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

72. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

75. The method of claim 11, wherein the memory comprises: a physical memory.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 27 above.	

76. The method of claim 11, wherein the operating system comprises: a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

77. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

79. The method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 31 above.	

80. The method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 32, 33 and 49 above.	

81. The method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form.	Mealey, as evidenced by the example citations below, discloses "the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 32, 33 and 49 above.	

83. The method of claim 13, wherein the boot data in the compressed form represents a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

84. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

87. The method of claim 13, wherein the memory comprises: a physical memory.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 27 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

88. The method of claim 13, wherein the operating system comprises: a plurality of files.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 16 and 23 above.	

89. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 15, 17 and 24 above.	

91. The method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 31 above.	

92. The method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 32, 33 and 49 above.	

Appendix C34
Invalidity of U.S. Patent 8,880,862 based on Mealey

93. The method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Mealey, as evidenced by the example citations below, discloses "the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claim 32, 33, and 49 above.	

97.1 The method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list,	Mealey, as evidenced by the example citations below, discloses "the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 9.1-9.3 and 19 above.	

97.2 and wherein the updating comprises: associating the additional compressed boot data with the boot data list.	Mealey, as evidenced by the example citations below, discloses "and wherein the updating comprises: associating the additional compressed boot data with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, and wherein the updating comprises: associating the additional compressed boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.4.1, 2, 4, 5.6, and 8.6 above.	

98. The method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list.	Mealey, as evidenced by the example citations below, discloses "the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

107. The method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.	Mealey, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

108. The method of claim 2, further comprising: compressing at least a portion of the additional boot data.	Mealey, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: compressing at least a portion of the additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: compressing at least a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 1.1, 5, 9.1 and 8 above.	

109. The method of claim 108, further comprising: storing the compressed additional boot data.	Mealey, as evidenced by the example citations below, discloses "the method of claim 108, further comprising: storing the compressed additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 108, further comprising: storing the compressed additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Mealey discloses this limitation: *See* Claims 5.1, 8.1 and 9.1 above.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

The publication Menon, A performance comparison of RAID-5 and log-structured arrays, IBM Almaden Research Center, ("Menon") invalidates claims 1-6, 8-11, 13-17, 19, 23-24, 27-29, 31-33, 35-36, 39-40, 43-45, 47-53, 59-60, 63-65, 67, 71-72, 75-77, 79-81, 83-84, 87-89, 91-93, 97-98, and 107-109 of United States Patent No. 8,880,862 ("the '862 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '862 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

1 (Preamble) A method for providing accelerated loading of an operating system in a computer system, the method comprising:	Menon, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, the method comprising:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system in a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "In addition to improved transfer times, performance benefits result from the fact that by storing compressed data in the subsystem cache, we get an effectively larger cache."

Menon, 167.

> "A related parameter is the compression ratio. Improved compression ratios help in two ways. First, the better the compression ratio the greater the free space and, hence, the better the performance. Second, the better the compression ratio, the better the cache hit Iatios. This second factor aEects performance quite significantly, particularly at low 1/0 rates where it has a bigger impact than the first factor."

Menon, 173.

> "LSA has an advantage over standard RAID-5 in that the data is compressed in the controller cache, so it appears to have an effectively larger cache. It is also possible to implement a version of RAID-5 in which data is stored compressed in cache. The approach would be as follows. When a record is written by the system, store it in compressed form in the cache. When the record is written to disk, write it in compressed form, but leave enough pad after it to be able to store the full uncompressed form of the record. This ensures that we can always do update-in-place. With this approach, no disk space is saved and :io LSA directory is needed, but improved performance or lower cost is possible because the data is stored compressed in cache. We call this approach ***RAID-5 with compression***."

Menon, 174.

> "In understanding these comparisons, keep in nind the following

assumptions: (1) the storing of old data for RAID- 5 and the LSA directory for LSA are both assurned to take 17% of the cache, so only 83?0 of the cache is used for other purposes (2) schemes that store compressed data in cache have an effectkcly larger cache (3) RAID-5 has flat skew within an array but has some skew between arrays, LSA has no skew within or across arrays (4) Only LSA gets better transfer times due to compressed data on disk (5) RAID-5 has better seek affinity than LSA." Menon, 175.

1.1 loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;	Menon, as evidenced by the example citations below, discloses "loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "In addition to improved transfer times, performance benefits result from the fact that by storing compressed data in the subsystem cache, we get an effectively larger cache."

Menon, 167.

> "In LSA, data is stored on disks in compressed form. After a piece of data is updated, it may not compress as well as it did before it was updated, so it may not fit back into the space that had been allocated for it bcfore the update."

Menon, 169.

> "The record is compressed as soon as it reaches the subsystem. Compressed records are stored in the controller cache."

Menon, 169.

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."

Menon, 169.

1.2 accessing the loaded portion of the boot data in the compressed form from memory;	Menon, as evidenced by the example citations below, discloses "accessing the loaded portion of the boot data in the compressed form from memory;"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the boot data in the compressed form from memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: "The record is compressed as soon as it reaches the subsystem. Compressed records are stored in the controller cache." Menon, 169. "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host." Menon, 169.	

1.3 decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and	Menon, as evidenced by the example citations below, discloses "decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

> "In addition to improved transfer times, performance benefits result from the fact that by storing compressed data in the subsystem cache, we get an effectively larger cache."

Menon, 167.

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."

Menon, 169.

> "A related parameter is the compression ratio. Improved compression ratios help in two ways. First, the better the compression ratio the greater the free space and, hence, the better the performance. Second, the better the compression ratio, the better the cache hit Iatios. This second factor aEects performance quite significantly, particularly at low 1/0 rates where it has a bigger impact than the first factor."

Menon, 173.

> "LSA has an advantage over standard RAID-5 in that the data is compressed in the controller cache, so it appears to have an effectively larger cache. It is also possible to implement a version of RAID-5 in which data is stored compressed in cache. The approach would be as

> follows. When a record is written by the system, store it in compressed form in the cache. When the record is written to disk, write it in compressed form, but leave enough pad after it to be able to store the full uncompressed form of the record. This ensures that we can always do update-in-place. With this approach, no disk space is saved and :io LSA directory is needed, but improved performance or lower cost is possible because the data is stored compressed in cache. We call this approach ***RAID-5 with compression***."

Menon, 174.

> "In understanding these comparisons, keep in nind the following assumptions: (1) the storing of old data for RAID- 5 and the LSA directory for LSA are both assurned to take 17% of the cache, so only 83?0 of the cache is used for other purposes (2) schemes that store compressed data in cache have an effectkcly larger cache (3) RAID-5 has flat skew within an array but has some skew between arrays, LSA has no skew within or across arrays (4) Only LSA gets better transfer times due to compressed data on disk (5) RAID-5 has better seek affinity than LSA."

Menon, 175.

> "Our results for IMS workloads may be summarized as follows. RAID-5 with compression has better response time and throughput than RAID-5 without compression, so we should try to do the former whenever possible."

Menon, 176.

1.4.1 updating the boot data list,	Menon, as evidenced by the example citations below, discloses "updating the boot data list,"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

1.4.2 wherein the decompressed portion of boot data comprises a portion of the operating system.	Menon, as evidenced by the example citations below, discloses "wherein the decompressed portion of boot data comprises a portion of the operating system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the decompressed portion of boot data comprises a portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

2. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list.	Menon, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

3. The method of claim 1, wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list.	Menon, as evidenced by the example citations below, discloses "wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein removing an association of additional boot data that is associated with the boot data list from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

4. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data.	Menon, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list; and compressing a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.4.1, and 2 above.	

5 (Preamble) A method for booting a computer system, the method comprising:	Menon, as evidenced by the example citations below, discloses "a method for booting a computer system, the method comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1-1.4.2 above.	

5.1 storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory;	Menon, as evidenced by the example citations below, discloses "storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

5.2 loading the stored compressed boot data from the first memory;	Menon, as evidenced by the example citations below, discloses "loading the stored compressed boot data from the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the stored compressed boot data from the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

5.3 accessing the loaded compressed boot data;	Menon, as evidenced by the example citations below, discloses "accessing the loaded compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.2 above.	

5.4 decompressing the accessed compressed boot data;	Menon, as evidenced by the example citations below, discloses "decompressing the accessed compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.3 above.	

5.5 utilizing the decompressed boot data to at least partially boot the computer system;	Menon, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to at least partially boot the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

See Claims 1.3 and 1.4.2 above.

See also

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."

Menon, 169.

5.6 updating the boot data list;	Menon, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

5.7 wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.	Menon, as evidenced by the example citations below, discloses "wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1-1.3 above.	

6 (Preamble) A system comprising: a processor;	Menon, as evidenced by the example citations below, discloses “a system comprising: a processor:”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system comprising: a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

6.1 a memory; and	Menon, as evidenced by the example citations below, discloses "a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.2 a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor	Menon, as evidenced by the example citations below, discloses "a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.3 wherein the processor is configured:	Menon, as evidenced by the example citations below, discloses “wherein the processor is configured:”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the processor is configured), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

6.4 to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory,	Menon, as evidenced by the example citations below, discloses "to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

6.5 to access the loaded portion of the boot data in the compressed form,	Menon, as evidenced by the example citations below, discloses "to access the loaded portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to access the loaded portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.2 above.	

6.6 to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data, and	Menon, as evidenced by the example citations below, discloses "to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.3 above.	

6.7 to update the boot data list.	Menon, as evidenced by the example citations below, discloses “to update the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to update the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

8 (Preamble) A method of loading an operating system for booting a computer system, comprising:	Menon, as evidenced by the example citations below, discloses "A method of loading an operating system for booting a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method of loading an operating system for booting a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1 (Preamble) above.	

8.1 storing a portion of the operating system in a compressed form in a first memory;	Menon, as evidenced by the example citations below, discloses “storing a portion of the operating system in a compressed form in a first memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing a portion of the operating system in a compressed form in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 6.2 above.	

8.2 loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list;	Menon, as evidenced by the example citations below, discloses “loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

8.3 accessing the loaded portion of the operating system from the second memory in the compressed form;	Menon, as evidenced by the example citations below, discloses "accessing the loaded portion of the operating system from the second memory in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the operating system from the second memory in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.2 above.	

8.4 decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system;	Menon, as evidenced by the example citations below, discloses "decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.5 utilizing the decompressed portion of the operating system to at least partially boot the computer system; and	Menon, as evidenced by the example citations below, discloses “utilizing the decompressed portion of the operating system to at least partially boot the computer system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed portion of the operating system to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.3 and 1.4.2 above. *See also* “When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host.” Menon, 169.	

8.6 updating the boot data list,	Menon, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

8.7 wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form.	Menon, as evidenced by the example citations below, discloses “wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 1-1.3 and 5.7 above.	

9.1 The method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list; and	Menon, as evidenced by the example citations below, discloses "the method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, compressing an additional portion of the operating system that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

9.2 storing the additional portion of the operating system in the first memory, and	Menon, as evidenced by the example citations below, discloses "storing the additional portion of the operating system in the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing the additional portion of the operating system in the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 8.1 above.	

9.3 wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system.	Menon, as evidenced by the example citations below, discloses "wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 8.5 above.	

10. The method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder.	Menon, as evidenced by the example citations below, discloses "the method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 9.1 above.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

11 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Menon, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1 (Preamble) above.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

11.1 loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system;	Menon, as evidenced by the example citations below, discloses “loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

11.2 accessing the loaded boot data in compressed form from the memory;	Menon, as evidenced by the example citations below, discloses "accessing the loaded boot data in compressed form from the memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in compressed form from the memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.2 above.	

11.3 decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Menon, as evidenced by the example citations below, discloses "decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.3 above.	

11.4 utilizing the decompressed boot data to load at least a portion of the operating system for the computer system; and	Menon, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to load at least a portion of the operating system for the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to load at least a portion of the operating system for the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* ***<u>Add disclosure from '936 Patent Claim 1.5</u>*** "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host." Menon, 169.	

11.5 updating the boot data list.	Menon, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

13 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Menon, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1 (Preamble) above.	

13.1 loading boot data in a compressed form that is associated with a boot data list from a boot device;	Menon, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

13.2 accessing the loaded boot data in the compressed form;	Menon, as evidenced by the example citations below, discloses "accessing the loaded boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.2 above.	

13.3 decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Menon, as evidenced by the example citations below, discloses "decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.3 above.	

13.4 updating the boot data list.	Menon, as evidenced by the example citations below, discloses “updating the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

14 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Menon, as evidenced by the example citations below, discloses “a method for providing accelerated loading of an operating system in a computer system, comprising”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 1 (Preamble) above.	

14.1 accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list;	Menon, as evidenced by the example citations below, discloses "accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1 and 1.2 above.	

14.2 loading the boot data into a memory; and	Menon, as evidenced by the example citations below, discloses "loading the boot data into a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the boot data into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.1 above.	

14.3 servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form; and	Menon, as evidenced by the example citations below, discloses "servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Menon discloses this limitation:

See Claims 1.2 and 1.3 above.

See also

> "In addition to improved transfer times, performance benefits result from the fact that by storing compressed data in the subsystem cache, we get an effectively larger cache."

Menon, 167.

> "When the host system tries to read a record, we fetch the entire logical track containing that record from disk. This entire track is stored in the controller cache; the requested record alone is decompressed and sent to the host."

Menon, 169.

> "A related parameter is the compression ratio. Improved compression ratios help in two ways. First, the better the compression ratio the greater the free space and, hence, the better the performance. Second, the better the compression ratio, the better the cache hit Iatios. This second factor aEects performance quite significantly, particularly at low 1/0 rates where

it has a bigger impact than the first factor."

Menon, 173.

> "LSA has an advantage over standard RAID-5 in that the data is compressed in the controller cache, so it appears to have an effectively larger cache. It is also possible to implement a version of RAID-5 in which data is stored compressed in cache. The approach would be as follows. When a record is written by the system, store it in compressed form in the cache. When the record is written to disk, write it in compressed form, but leave enough pad after it to be able to store the full uncompressed form of the record. This ensures that we can always do update-in-place. With this approach, no disk space is saved and :io LSA directory is needed, but improved performance or lower cost is possible because the data is stored compressed in cache. We call this approach ***RAID-5 with compression***."

Menon, 174.

> "In understanding these comparisons, keep in nind the following assumptions: (1) the storing of old data for RAID- 5 and the LSA directory for LSA are both assurned to take 17% of the cache, so only 83?0 of the cache is used for other purposes (2) schemes that store compressed data in cache have an effectkcly larger cache (3) RAID-5 has flat skew within an array but has some skew between arrays, LSA has no skew within or across arrays (4) Only LSA gets better transfer times due to compressed data on disk (5) RAID-5 has better seek affinity than LSA."

Menon, 175.

> "Our results for IMS workloads may be summarized as follows. RAID-5 with compression has better response time and throughput than RAID-5 without compression, so we should try to do the former whenever possible."

Menon, 176.

14.4 updating the boot data list.	Menon, as evidenced by the example citations below, discloses “updating the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 1.4.1 above.	

15. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system.	Menon, as evidenced by the example citations below, discloses “the method of claim 14, wherein the boot data comprises: a program code associated with the operating system.”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Menon discloses this limitation:

See Claims 1 (Preamble) and 1.1 above.

16. The method of claim 14, wherein the operating system comprises: a plurality of files.	Menon, as evidenced by the example citations below, discloses “the method of claim 14, wherein the operating system comprises: a plurality of files.”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

17. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.	Menon, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 15 above.	

19.1 The method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:	Menon, as evidenced by the example citations below, discloses "the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

19.2 associating the accessed boot data that is not associated with the boot data list to the boot data list.	Menon, as evidenced by the example citations below, discloses "associating the accessed boot data that is not associated with the boot data list to the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, associating the accessed boot data that is not associated with the boot data list to the boot data list.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 2 above.	

23. The method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files.	Menon, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 16 above.	

24. The method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.	Menon, as evidenced by the example citations below, discloses “the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 15 above.	

27. The method of claim 1, wherein the memory comprises: a physical memory.	Menon, as evidenced by the example citations below, discloses "the method of claim 1, wherein the memory comprises: a physical memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1 and 1.2 above.	

28. The method of claim 1, wherein the operating system comprises: a plurality of files.	Menon, as evidenced by the example citations below, discloses “the method of claim 1, wherein the operating system comprises: a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 16 above.	

29. The method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program.	Menon, as evidenced by the example citations below, discloses “the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 15 and 17 above.	

31. The method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access.	Menon, as evidenced by the example citations below, discloses "the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1 and 1.2 above.	

32. The method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form.	Menon, as evidenced by the example citations below, discloses "the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

33. The method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form.	Menon, as evidenced by the example citations below, discloses "the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

35. The method of claim 5, wherein the compressed boot data represents a plurality of files.	Menon, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

36. The method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system.	Menon, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 15 and 17 above.	

39. The method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory.	Menon, as evidenced by the example citations below, discloses "the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1 and 1.2 above.	

40. The method of claim 36, wherein the operating system comprises: a plurality of files.	Menon, as evidenced by the example citations below, discloses "the method of claim 36, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 36, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

43. The method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access.	Menon, as evidenced by the example citations below, discloses "the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 31 above.	

44. The method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Menon, as evidenced by the example citations below, discloses “the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 32 above.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

45. The method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Menon, as evidenced by the example citations below, discloses "the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 32 and 33 above.	

47. The system of claim 6, wherein the boot data in the compressed form represents a plurality of files.	Menon, as evidenced by the example citations below, discloses “the system of claim 6, wherein the boot data in the compressed form represents a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

48. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system.	Menon, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

49. The system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form.	Menon, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 10 above.	

50. The system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form.	Menon, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

51. The system of claim 6, wherein the first memory comprises: a physical memory.	Menon, as evidenced by the example citations below, discloses "the system of claim 6, wherein the first memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the first memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 27 and 39 above.	

52. The system of claim 6, wherein the boot data the compressed form comprises: a plurality of files.	Menon, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

53. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program.	Menon, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

59. The method of claim 8, wherein the operating system in the compressed form represents a plurality of files.	Menon, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

60. The method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system.	Menon, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

63. The method of claim 8, wherein the second memory comprises: a physical memory.	Menon, as evidenced by the example citations below, discloses “the method of claim 8, wherein the second memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 27 and 38 above.	

64. The method of claim 8, wherein the operating system comprises: a plurality of files.	Menon, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

65. The method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program.	Menon, as evidenced by the example citations below, discloses “the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

67. The method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access.	Menon, as evidenced by the example citations below, discloses "the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 31 above.	

Appendix C35
Invalidity of U.S. Patent 8,880,862 based on Menon

71. The method of claim 11, wherein the boot data in the compressed form represents a plurality of files.	Menon, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

72. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Menon, as evidenced by the example citations below, discloses “the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

75. The method of claim 11, wherein the memory comprises: a physical memory.	Menon, as evidenced by the example citations below, discloses "the method of claim 11, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 27 above.	

76. The method of claim 11, wherein the operating system comprises: a plurality of files.	Menon, as evidenced by the example citations below, discloses “the method of claim 11, wherein the operating system comprises: a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

77. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program.	Menon, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

79. The method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access.	Menon, as evidenced by the example citations below, discloses "the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 31 above.	

80. The method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form.	Menon, as evidenced by the example citations below, discloses “the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 32, 33 and 49 above.	

81. The method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form.	Menon, as evidenced by the example citations below, discloses "the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 32, 33 and 49 above.	

83. The method of claim 13, wherein the boot data in the compressed form represents a plurality of files.	Menon, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

84. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Menon, as evidenced by the example citations below, discloses “the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

87. The method of claim 13, wherein the memory comprises: a physical memory.	Menon, as evidenced by the example citations below, discloses “the method of claim 13, wherein the memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claim 27 above.	

88. The method of claim 13, wherein the operating system comprises: a plurality of files.	Menon, as evidenced by the example citations below, discloses “the method of claim 13, wherein the operating system comprises: a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 16 and 23 above.	

89. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program.	Menon, as evidenced by the example citations below, discloses “the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 15, 17 and 24 above.	

91. The method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access.	Menon, as evidenced by the example citations below, discloses "the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 31 above.	

92. The method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Menon, as evidenced by the example citations below, discloses "the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 32, 33 and 49 above.	

93. The method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Menon, as evidenced by the example citations below, discloses "the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claim 32, 33, and 49 above.	

97.1 The method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list,	Menon, as evidenced by the example citations below, discloses “the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Menon discloses this limitation: *See* Claims 9.1-9.3 and 19 above.	

97.2 and wherein the updating comprises: associating the additional compressed boot data with the boot data list.	Menon, as evidenced by the example citations below, discloses "and wherein the updating comprises: associating the additional compressed boot data with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, and wherein the updating comprises: associating the additional compressed boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.4.1, 2, 4, 5.6, and 8.6 above.	

98. The method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list.	Menon, as evidenced by the example citations below, discloses "the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

107. The method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.	Menon, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

108. The method of claim 2, further comprising: compressing at least a portion of the additional boot data.	Menon, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: compressing at least a portion of the additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: compressing at least a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 1.1, 5, 9.1 and 8 above.	

109. The method of claim 108, further comprising: storing the compressed additional boot data.	Menon, as evidenced by the example citations below, discloses "the method of claim 108, further comprising: storing the compressed additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 108, further comprising: storing the compressed additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Menon discloses this limitation: *See* Claims 5.1, 8.1 and 9.1 above.	

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

The publication Rubini, Booting the Kernel, Linux Journal, Jan. 1997, ("Rubini") invalidates claims 1-6, 8-11, 13-17, 19, 23-24, 27-29, 31-33, 35-36, 39-40, 43-45, 47-53, 59-60, 63-65, 67, 71-72, 75-77, 79-81, 83-84, 87-89, 91-93, 97-98, and 107-109 of United States Patent No. 8,880,862 ("the '862 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '862 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

1 (Preamble) A method for providing accelerated loading of an operating system in a computer system, the method comprising:	Rubini, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, the method comprising:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system in a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

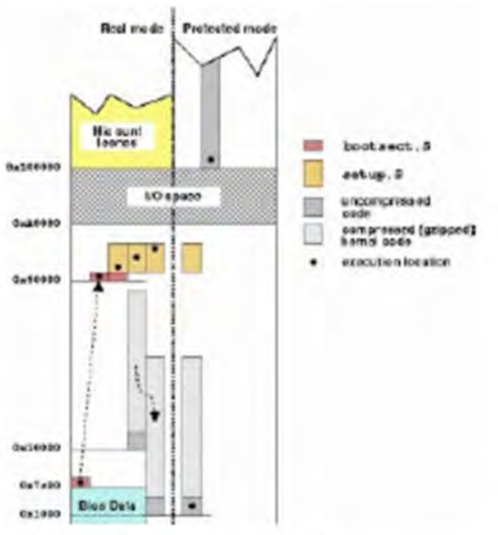


Figure 1. System Boot Data Map

Fig. 1.

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

Rubini, 1.

1.1 loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;	Rubini, as evidenced by the example citations below, discloses "loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as "real mode" and is dictated by compatibility with older processors of the same family."

Rubini, 1.

> "To make things difficult, the PC firmware loads only half a kilobyte of code and establishes its own memory layout before loading this first sector. Whatever the boot media, the first sector of the boot partition is loaded into memory at the address 0x7c00, where execution begins. What happens at 0x7c00 depends on the boot loader being used; we examine three situations here: no boot-loader, LILO, Loadlin."

Rubini, 2.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some

other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.

The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.

Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The boot steps shown above rely on the assumption that the compressed kernel can fit in half a megabyte of space. While this is true most of the time, a system stuffed with device drivers might not fit into this space. For example, kernels used in installation disks can easily outgrow the available space. Some new method is needed to solve the problem—this new method is called bzImage and was introduced in kernel version 1.3.73.

A bzImage is generated by issuing make bzImage from the top level Linux source directory. This kind of kernel image boots similarly to zImage, with a few changes:

- When the system is loaded to address 0x10000, a little helper routine is called after loading each 64K data block. The helper routine moves the data block to high memory by using a special BIOS call. Only the newer BIOS versions implement this functionality, and so, make boot still builds the conventional zImage, though this may change in the near future.
- setup.S doesn't move the system back to 0x1000 (4K) but, after entering protected mode, jumps instead directly to address 0x100000 (1MB) where data has been moved by the BIOS in the previous step."

Rubini, 3-4.

> "The rule for building the big compressed image can be read from Makefile; it affects several files in arch/i386/boot. One good point of bzImage is that when kernel/head.S is called, it doesn't notice the extra work, and everything goes forward as usual."

Rubini, 4.

1.2 accessing the loaded portion of the boot data in the compressed form from memory;	Rubini, as evidenced by the example citations below, discloses "accessing the loaded portion of the boot data in the compressed form from memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the boot data in the compressed form from memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.
>
> - The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using

the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.

- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the "gunzip" part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of "gunzipping" the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the "real" mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C."

Rubini, 3.

"The boot steps shown above rely on the assumption that the compressed kernel can fit in half a megabyte of space. While this is true most of the time, a system stuffed with device drivers might not fit into this space. For example, kernels used in installation disks can easily outgrow the available space. Some new method is needed to solve the problem—this new method is called bzImage and was introduced in kernel version 1.3.73.

A bzImage is generated by issuing make bzImage from the top level Linux source directory. This kind of kernel image boots similarly to zImage, with a few changes:

- When the system is loaded to address 0x10000, a little helper routine is called after loading each 64K data block. The helper routine moves the data block to high memory by using a special BIOS call. Only the newer BIOS versions implement this functionality, and so, make boot still builds the conventional zImage, though this may change in the near future.
- setup.S doesn't move the system back to 0x1000 (4K) but, after entering protected mode, jumps instead directly to address 0x100000 (1MB) where data has been moved by the BIOS in the previous step."

Rubini, 3-4.

"The rule for building the big compressed image can be read from Makefile; it affects several files in arch/i386/boot. One good point of bzImage is that when kernel/head.S is called, it doesn't notice the extra work, and everything goes forward as usual."

Rubini, 4.

1.3 decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and	Rubini, as evidenced by the example citations below, discloses "decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path

names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the "gunzip" part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of "gunzipping" the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the "real" mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C."

Rubini, 3. "The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly." Rubini, 4.

1.4.1 updating the boot data list,	Rubini, as evidenced by the example citations below, discloses "updating the boot data list,"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: "Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory." Rubini, 2.	

1.4.2 wherein the decompressed portion of boot data comprises a portion of the operating system.	Rubini, as evidenced by the example citations below, discloses "wherein the decompressed portion of boot data comprises a portion of the operating system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the decompressed portion of boot data comprises a portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called “zero-page”, used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run.”

Rubini, 2-3.

“The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the “gunzip” part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of “gunzipping” the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the “real” mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C.”

Rubini, 3.

"The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly."   Rubini, 4.

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

2. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list.	Rubini, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

3. The method of claim 1, wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list.	Rubini, as evidenced by the example citations below, discloses "wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein removing an association of additional boot data that is associated with the boot data list from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

4. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data.	Rubini, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list; and compressing a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.4.1, and 2 above.	

5 (Preamble) A method for booting a computer system, the method comprising:	Rubini, as evidenced by the example citations below, discloses "a method for booting a computer system, the method comprising:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claims 1-1.4.2 above.

See also **Add disclosure from '608 Patent Claims 1.1 and 1.2.**

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory."

Rubini, 2.

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while

others call it “console” to focus on user interaction.

The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system.”

Rubini, 1.

“When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as “real mode” and is dictated by compatibility with older processors of the same family.”

Rubini, 1.

“Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.”

Rubini, 2.

5.1 storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory;	Rubini, as evidenced by the example citations below, discloses "storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above. *See also* "The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory." Rubini, 2.	

5.2 loading the stored compressed boot data from the first memory;	Rubini, as evidenced by the example citations below, discloses “loading the stored compressed boot data from the first memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the stored compressed boot data from the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

5.3 accessing the loaded compressed boot data;	Rubini, as evidenced by the example citations below, discloses "accessing the loaded compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.2 above.	

5.4 decompressing the accessed compressed boot data;	Rubini, as evidenced by the example citations below, discloses "decompressing the accessed compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.3 above.	

5.5 utilizing the decompressed boot data to at least partially boot the computer system;	Rubini, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to at least partially boot the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claims 1.3 and 1.4.2 above.

See also

> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path

names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the "gunzip" part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of "gunzipping" the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the "real" mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C."

Rubini, 3.

> "The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly."

Rubini, 4.

5.6 updating the boot data list;	Rubini, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

5.7 wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.	Rubini, as evidenced by the example citations below, discloses "wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1-1.3 above.	

6 (Preamble) A system comprising: a processor;	Rubini, as evidenced by the example citations below, discloses “a system comprising: a processor:”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system comprising: a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Rubini discloses this limitation:

> “In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is “unmovable software” found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or “flash” to stress its hardware implementation, while others call it “console” to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system.”

Rubini, 1.

> “When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as “real mode” and is dictated by compatibility with older processors of the same family.”

Rubini, 1.

> “Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.”

Rubini, 2.

6.1 a memory; and	Rubini, as evidenced by the example citations below, discloses “a memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.2 a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor	Rubini, as evidenced by the example citations below, discloses "a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claims 1.1 and 1.2 above.

See also

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as "real mode" and is dictated by compatibility with older processors of the same family."

Rubini, 1.

> "Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards."

Rubini, 2.

6.3 wherein the processor is configured:	Rubini, as evidenced by the example citations below, discloses "wherein the processor is configured:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the processor is configured), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See also

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as "real mode" and is dictated by compatibility with older processors of the same family."

Rubini, 1.

> "Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards."

Rubini, 2.

6.4 to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory,	Rubini, as evidenced by the example citations below, discloses “to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

6.5 to access the loaded portion of the boot data in the compressed form,	Rubini, as evidenced by the example citations below, discloses "to access the loaded portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to access the loaded portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.2 above.	

6.6 to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data, and	Rubini, as evidenced by the example citations below, discloses “to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.3 above.	

6.7 to update the boot data list.	Rubini, as evidenced by the example citations below, discloses "to update the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to update the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

8 (Preamble) A method of loading an operating system for booting a computer system, comprising:	Rubini, as evidenced by the example citations below, discloses “A method of loading an operating system for booting a computer system, comprising:”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method of loading an operating system for booting a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1 (Preamble) above.	

8.1 storing a portion of the operating system in a compressed form in a first memory;	Rubini, as evidenced by the example citations below, discloses "storing a portion of the operating system in a compressed form in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing a portion of the operating system in a compressed form in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 6.2 above.	

8.2 loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list;	Rubini, as evidenced by the example citations below, discloses “loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

8.3 accessing the loaded portion of the operating system from the second memory in the compressed form;	Rubini, as evidenced by the example citations below, discloses "accessing the loaded portion of the operating system from the second memory in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the operating system from the second memory in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.2 above.	

8.4 decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system;	Rubini, as evidenced by the example citations below, discloses "decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.5 utilizing the decompressed portion of the operating system to at least partially boot the computer system; and	Rubini, as evidenced by the example citations below, discloses "utilizing the decompressed portion of the operating system to at least partially boot the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed portion of the operating system to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claims 1.3 and 1.4.2 above.

See also

> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low

640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the "gunzip" part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of "gunzipping" the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the "real" mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed

> boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C."

Rubini, 3.

> "The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly."

Rubini, 4.

8.6 updating the boot data list,	Rubini, as evidenced by the example citations below, discloses “updating the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

8.7 wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form.	Rubini, as evidenced by the example citations below, discloses "wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1-1.3 and 5.7 above.	

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

9.1 The method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list; and	Rubini, as evidenced by the example citations below, discloses "the method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, compressing an additional portion of the operating system that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

9.2 storing the additional portion of the operating system in the first memory, and	Rubini, as evidenced by the example citations below, discloses "storing the additional portion of the operating system in the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing the additional portion of the operating system in the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 8.1 above.	

9.3 wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system.	Rubini, as evidenced by the example citations below, discloses "wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 8.5 above.	

10. The method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder.	Rubini, as evidenced by the example citations below, discloses "the method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 9.1 above.	

11 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Rubini, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1 (Preamble) above.	

11.1 loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system;	Rubini, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claim 1.1 above.

See also

> "In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is "unmovable software" found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or "flash" to stress its hardware implementation, while others call it "console" to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "When the x86 processor is turned on, it is a 16-bit processor that sees only 1MB of RAM. This environment is known as "real mode" and is dictated by compatibility with older processors of the same family."

Rubini, 1.

> "Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards."

Rubini, 2.

11.2 accessing the loaded boot data in compressed form from the memory;	Rubini, as evidenced by the example citations below, discloses "accessing the loaded boot data in compressed form from the memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in compressed form from the memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.2 above.	

11.3 decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Rubini, as evidenced by the example citations below, discloses "decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.3 above.	

11.4 utilizing the decompressed boot data to load at least a portion of the operating system for the computer system; and	Rubini, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to load at least a portion of the operating system for the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to load at least a portion of the operating system for the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

> "The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system."

Rubini, 1.

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.

- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called “zero-page”, used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run.”

Rubini, 2-3.

“The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the “gunzip” part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of “gunzipping” the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the “real” mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C.”

Rubini, 3.

> "The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly."

Rubini, 4.

11.5 updating the boot data list.	Rubini, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

13 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Rubini, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1 (Preamble) above.	

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

13.1 loading boot data in a compressed form that is associated with a boot data list from a boot device;	Rubini, as evidenced by the example citations below, discloses “loading boot data in a compressed form that is associated with a boot data list from a boot device”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

13.2 accessing the loaded boot data in the compressed form;	Rubini, as evidenced by the example citations below, discloses "accessing the loaded boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.2 above.	

13.3 decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Rubini, as evidenced by the example citations below, discloses "decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.3 above.	

13.4 updating the boot data list.	Rubini, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

14 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Rubini, as evidenced by the example citations below, discloses “a method for providing accelerated loading of an operating system in a computer system, comprising”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1 (Preamble) above.	

14.1 accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list;	Rubini, as evidenced by the example citations below, discloses “accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Rubini discloses this limitation:

See Claims 1.1 and 1.2 above.

See also

> “In order to be able to use the computer when the power is turned on, the processor begins execution from the system's firmware. The firmware is “unmovable software” found in ROM; some manufacturers call it the Basic Input-Output System (BIOS) to underline its software role, some call it PROM or “flash” to stress its hardware implementation, while others call it “console” to focus on user interaction.
>
> The firmware usually checks the hardware's functionality, retrieves part (or all) of the kernel from a storage medium and executes it. This first part of the kernel must load the rest of itself and initialize the whole system.”

Rubini, 1.

> “The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a “big zImage” instead, the kernel file created is called bzImage and resides in the same directory.”

Rubini, 2.

14.2 loading the boot data into a memory; and	Rubini, as evidenced by the example citations below, discloses "loading the boot data into a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the boot data into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.1 above.	

14.3 servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form; and	Rubini, as evidenced by the example citations below, discloses "servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claims 1.2 and 1.3 above.

See also

> "Booting zImage and bzImage
>
> Even though it's rare to boot the system without a boot loader, it is still possible to do so by copying the raw kernel to a floppy disk. The command cat zImage >/dev/fd0 works perfectly on Linux, although some other Unix systems can do the task reliably only by using the dd command. Without going into detail, the raw floppy image created by zImage can then be configured using the rdev program.
>
> The file called zImage is the compressed kernel image that resides in arch/i386/boot after either make zImage or make boot is executed—the latter invocation is the one I prefer, as it works unchanged on other platforms. If you built a "big zImage" instead, the kernel file created is called bzImage and resides in the same directory.
>
> Booting an x86 kernel is a tricky task because of the limited amount of available memory. The Linux kernel tries to maximize usage of the low 640 kilobytes by moving itself around several times. Let's look at the

steps performed by a zImage kernel in detail; all of the following path names are relative to the arch/i386/boot directory.

- The first sector (executing at 0x7c00) moves itself to 0x90000 and loads subsequent sectors after itself, getting them from the boot device using the firmware's functions to access the disk. The rest of the kernel is then loaded to address 0x10000, allowing for a maximum size of half a megabyte of data—remember, this is the compressed image. The boot sector code lives in bootsect.S, a real-mode assembly file.
- Then code at 0x90200 (defined in setup.S) takes care of some hardware initialization and allows the default text mode (video.S) to be changed. Text mode selection is a compile-time option from 2.1.9 onwards.
- Later, all the kernel is moved from 0x10000 (64K) to 0x1000 (4K). This move overwrites BIOS data stored in RAM, so BIOS calls can no longer be performed. The first physical page is not touched because it is the so-called "zero-page", used in handling virtual memory.
- At this point, setup.S enters protected mode and jumps to 0x1000, where the kernel lives. All the available memory can be accessed now, and the system can begin to run."

Rubini, 2-3.

"The steps just described were once the whole story of booting when the kernel was small enough to fit in half a megabyte of memory—the address range between 0x10000 and 0x90000. As features were added to the system, the kernel became larger than half a megabyte and could no longer be moved to 0x1000. Thus, code at 0x1000 is no longer th Linux kernel, instead the "gunzip" part of the gzip program resides at that address. The following additional steps are now needed t uncompress the kernel and execute it:

- head.S in the compressed directory is at 0x1000, and is in charge of "gunzipping" the kernel; it calls the function decompress_kernel, defined in compressed/misc.c, which in turns calls inflate which writes its output starting at address 0x100000 (1MB). High memory can now be accessed, because the processor is definitely out of its limited boot environment—the "real" mode.
- After decompression, head.S jumps to the actual beginning of the kernel. The relevant code is in ../kernel/head.S, outside of the boot directory.

The boot process is now over, and head.S (i.e., the code found at 0x100000 that used to be at 0x1000 before introducing compressed

> boots) can complete processor initialization and call start_kernel(). Code for all functions after this step is written in C."

Rubini, 3.

> "The decompresser found at 1MB writes the uncompressed kernel image into low memory until it is exhausted, and then into high memory after the compressed image. The two pieces are then reassembled to the address 0x100000 (1MB). Several memory moves are needed to perform the task correctly."

Rubini, 4.

14.4 updating the boot data list.	Rubini, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 1.4.1 above.	

15. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system.	Rubini, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Rubini discloses this limitation:

See Claims 1 (Preamble) and 1.1 above.

See also

> "This article is a description of the steps required to boot the Linux kernel."

Rubini, 1.

> "When the power is turned on, however, the system software must boot the kernel and work in a limited operating environment."

Rubini, 1.

16. The method of claim 14, wherein the operating system comprises: a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 14, wherein the operating system comprises: a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: "A computer system is a complex machine, and the operating system is an elaborate tool that orchestrates hardware complexities to show a simple and standardized environment to the end user." Rubini, 1.	

17. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.	Rubini, as evidenced by the example citations below, discloses “the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claim 15 above.	

19.1 The method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:	Rubini, as evidenced by the example citations below, discloses "the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

19.2 associating the accessed boot data that is not associated with the boot data list to the boot data list.	Rubini, as evidenced by the example citations below, discloses "associating the accessed boot data that is not associated with the boot data list to the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, associating the accessed boot data that is not associated with the boot data list to the boot data list.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 2 above.	

23. The method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 16 above.	

24. The method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.	Rubini, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 15 above.	

27. The method of claim 1, wherein the memory comprises: a physical memory.	Rubini, as evidenced by the example citations below, discloses “the method of claim 1, wherein the memory comprises: a physical memory.”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.	

28. The method of claim 1, wherein the operating system comprises: a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 1, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 16 above.	

29. The method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program.	Rubini, as evidenced by the example citations below, discloses "the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15 and 17 above.	

31. The method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access.	Rubini, as evidenced by the example citations below, discloses “the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1 and 1.2 above.	

32. The method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form.	Rubini, as evidenced by the example citations below, discloses "the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

33. The method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form.	Rubini, as evidenced by the example citations below, discloses "the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

35. The method of claim 5, wherein the compressed boot data represents a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

Appendix C36
Invalidity of U.S. Patent 8,880,862 based on Rubini

36. The method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system.	Rubini, as evidenced by the example citations below, discloses “the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15 and 17 above.	

39. The method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory.	Rubini, as evidenced by the example citations below, discloses “the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1 and 1.2 above.	

40. The method of claim 36, wherein the operating system comprises: a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 36, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 36, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

43. The method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access.	Rubini, as evidenced by the example citations below, discloses "the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 31 above.	

44. The method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Rubini, as evidenced by the example citations below, discloses "the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 32 above.	

45. The method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Rubini, as evidenced by the example citations below, discloses "the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 32 and 33 above.	

47. The system of claim 6, wherein the boot data in the compressed form represents a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

48. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system.	Rubini, as evidenced by the example citations below, discloses “the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

49. The system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form.	Rubini, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 10 above.	

50. The system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form.	Rubini, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.	

51. The system of claim 6, wherein the first memory comprises: a physical memory.	Rubini, as evidenced by the example citations below, discloses "the system of claim 6, wherein the first memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the first memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 27 and 39 above.	

52. The system of claim 6, wherein the boot data the compressed form comprises: a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

53. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program.	Rubini, as evidenced by the example citations below, discloses “the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

59. The method of claim 8, wherein the operating system in the compressed form represents a plurality of files.	Rubini, as evidenced by the example citations below, discloses “the method of claim 8, wherein the operating system in the compressed form represents a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

60. The method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system.	Rubini, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

63. The method of claim 8, wherein the second memory comprises: a physical memory.	Rubini, as evidenced by the example citations below, discloses "the method of claim 8, wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 27 and 38 above.	

64. The method of claim 8, wherein the operating system comprises: a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

65. The method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program.	Rubini, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

67. The method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access.	Rubini, as evidenced by the example citations below, discloses "the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 31 above.	

71. The method of claim 11, wherein the boot data in the compressed form represents a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

72. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Rubini, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

75. The method of claim 11, wherein the memory comprises: a physical memory.	Rubini, as evidenced by the example citations below, discloses "the method of claim 11, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 27 above.	

76. The method of claim 11, wherein the operating system comprises: a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 11, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

77. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program.	Rubini, as evidenced by the example citations below, discloses “the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

79. The method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access.	Rubini, as evidenced by the example citations below, discloses "the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 31 above.	

80. The method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form.	Rubini, as evidenced by the example citations below, discloses "the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 32, 33 and 49 above.	

81. The method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form.	Rubini, as evidenced by the example citations below, discloses "the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 32, 33 and 49 above.	

83. The method of claim 13, wherein the boot data in the compressed form represents a plurality of files.	Rubini, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

84. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Rubini, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

87. The method of claim 13, wherein the memory comprises: a physical memory.	Rubini, as evidenced by the example citations below, discloses "the method of claim 13, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 27 above.	

88. The method of claim 13, wherein the operating system comprises: a plurality of files.	Rubini, as evidenced by the example citations below, discloses “the method of claim 13, wherein the operating system comprises: a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Rubini discloses this limitation: *See* Claims 16 and 23 above.	

89. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program.	Rubini, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 15, 17 and 24 above.	

91. The method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access.	Rubini, as evidenced by the example citations below, discloses "the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 31 above.	

92. The method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Rubini, as evidenced by the example citations below, discloses "the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 32, 33 and 49 above.	

93. The method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Rubini, as evidenced by the example citations below, discloses "the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claim 32, 33, and 49 above.	

97.1 The method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list,	Rubini, as evidenced by the example citations below, discloses "the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 9.1-9.3 and 19 above.	

97.2 and wherein the updating comprises: associating the additional compressed boot data with the boot data list.	Rubini, as evidenced by the example citations below, discloses "and wherein the updating comprises: associating the additional compressed boot data with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, and wherein the updating comprises: associating the additional compressed boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.4.1, 2, 4, 5.6, and 8.6 above.	

98. The method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list.	Rubini, as evidenced by the example citations below, discloses "the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

107. The method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.	Rubini, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

108. The method of claim 2, further comprising: compressing at least a portion of the additional boot data.	Rubini, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: compressing at least a portion of the additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: compressing at least a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 1.1, 5, 9.1 and 8 above.	

109. The method of claim 108, further comprising: storing the compressed additional boot data.	Rubini, as evidenced by the example citations below, discloses "the method of claim 108, further comprising: storing the compressed additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 108, further comprising: storing the compressed additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Rubini discloses this limitation: *See* Claims 5.1, 8.1 and 9.1 above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

Wynn, et al. "The effect of compression on performance in a demand paging operating system," The Journal of Systems and Software (2000) ("Wynn Article") and Wynn, "The Effect of Compression on Performance in a Demand Paging Operating System," 1997 ("Wynn Thesis") (collectively, "Wynn"), alone or in combination, invalidate claims 1-6, 8-11, 13-17, 19, 23-24, 27-29, 31-33, 35-36, 39-40, 43-45, 47-53, 59-60, 63-65, 67, 71-72, 75-77, 79-81, 83-84, 87-89, 91-93, 97-98, and 107-109 of United States Patent No. 8,880,862 ("the '862 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art, and/or in combination with the knowledge of a person of ordinary skill.

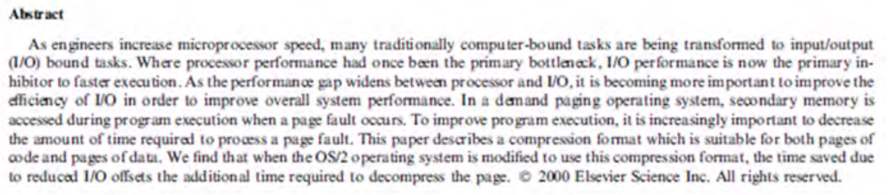
The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '862 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

1 (Preamble) A method for providing accelerated loading of an operating system in a computer system, the method comprising:	Wynn, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, the method comprising:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system in a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

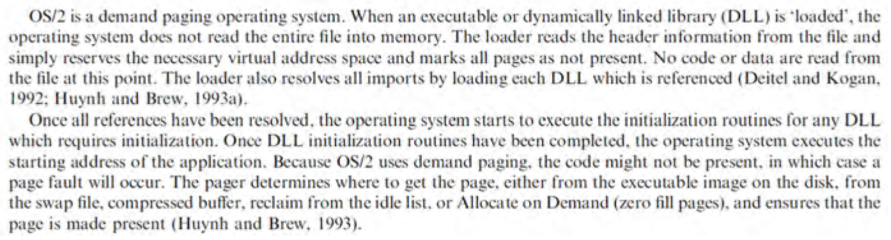
Wynn discloses this limitation:



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

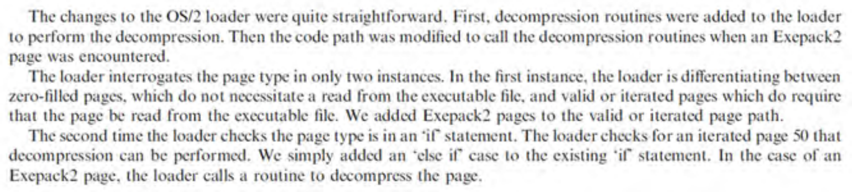
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

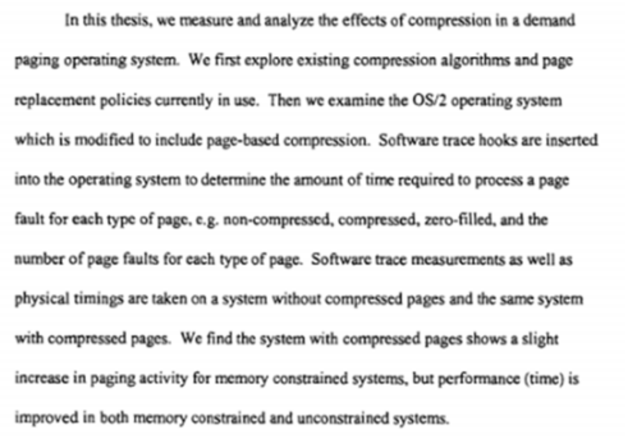


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

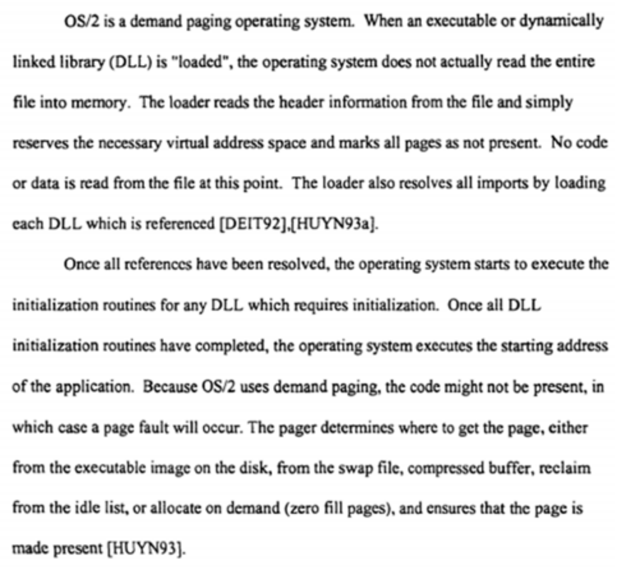
The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

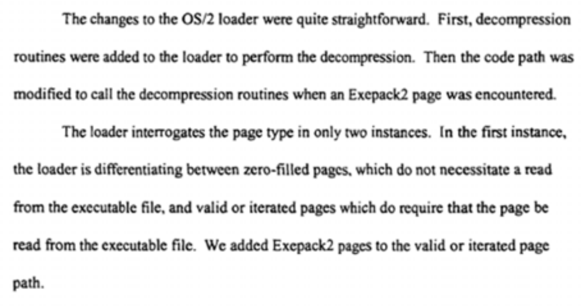
Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2



The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

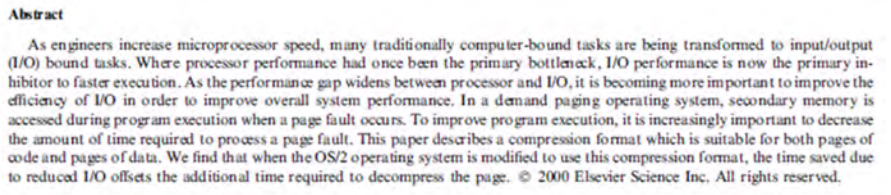
The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

1.1 loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;	Wynn, as evidenced by the example citations below, discloses “loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

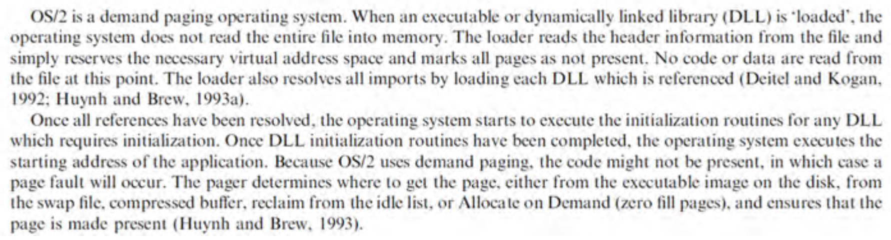
Wynn discloses this limitation:



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

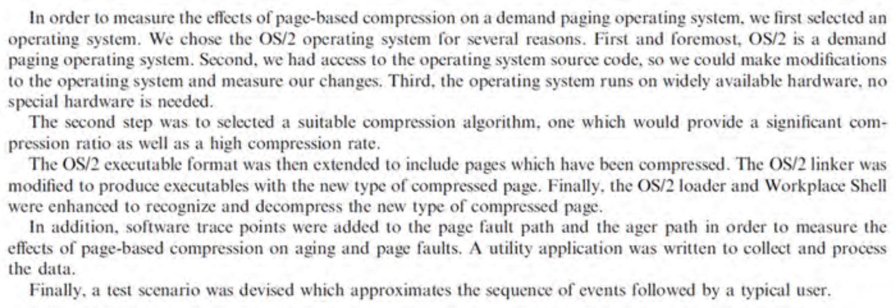
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is ‘loaded’, the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

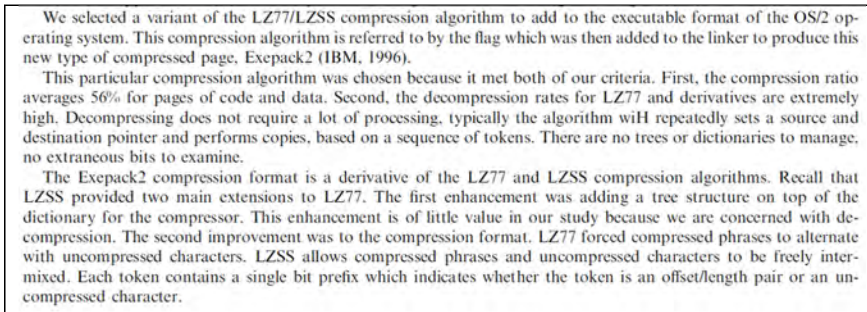
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

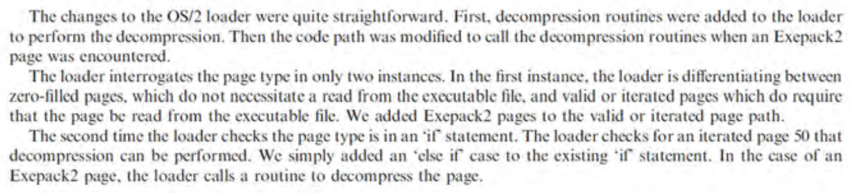


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

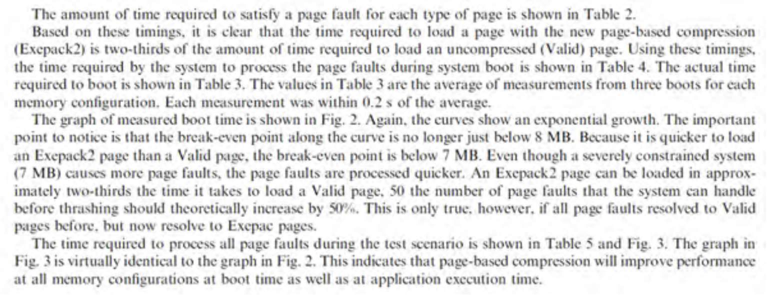


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4



The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

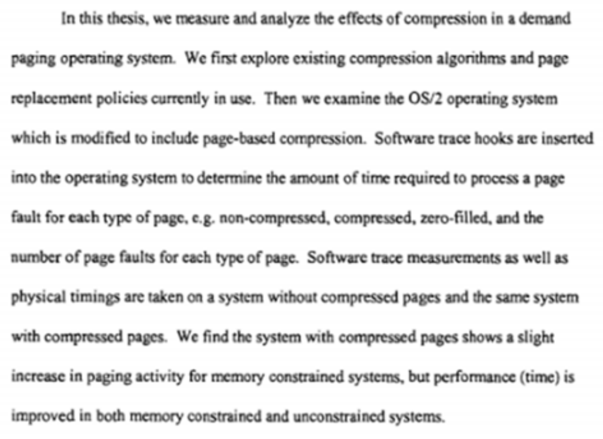
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

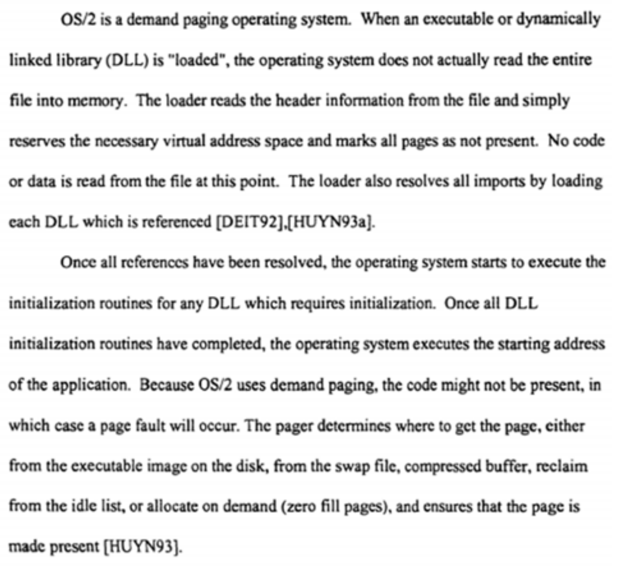
Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

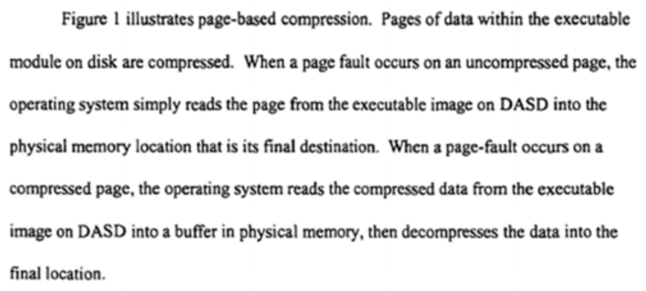
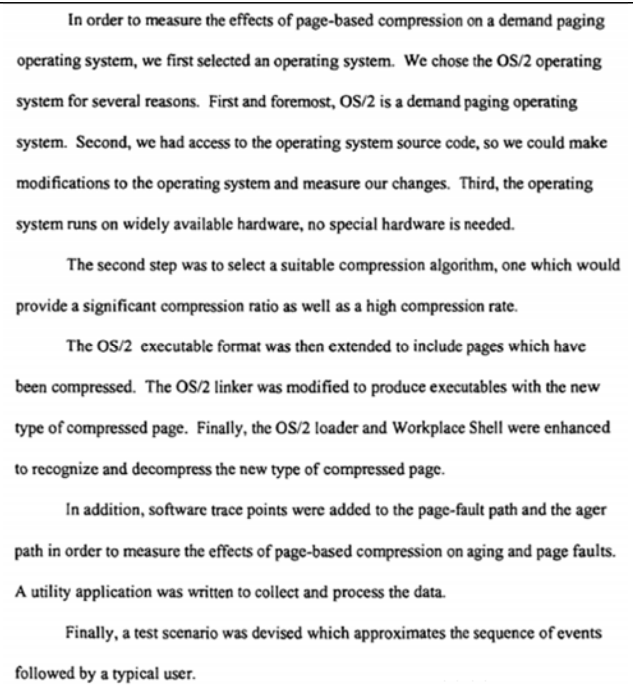


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

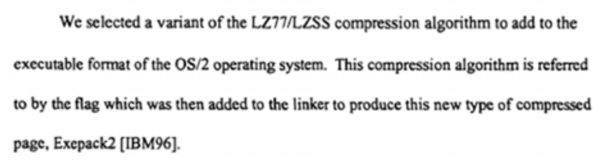
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

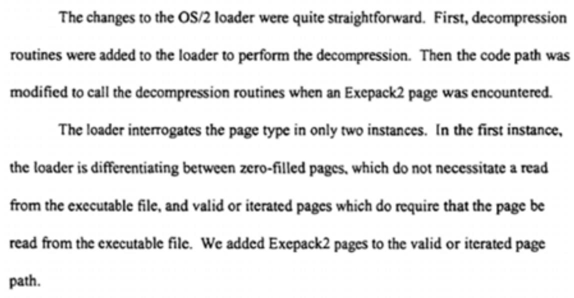
In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

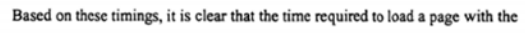

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

Wynn Thesis, 5.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5


Based on these timings, it is clear that the time required to load a page with the

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3

An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

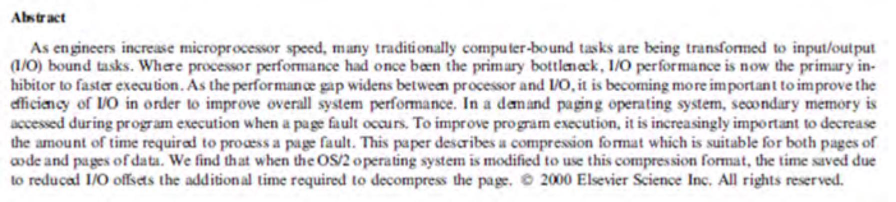
Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

1.2 accessing the loaded portion of the boot data in the compressed form from memory;	Wynn, as evidenced by the example citations below, discloses "accessing the loaded portion of the boot data in the compressed form from memory;"

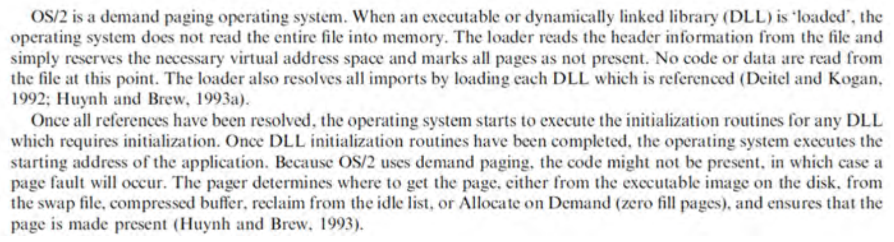
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the boot data in the compressed form from memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:


Abstract

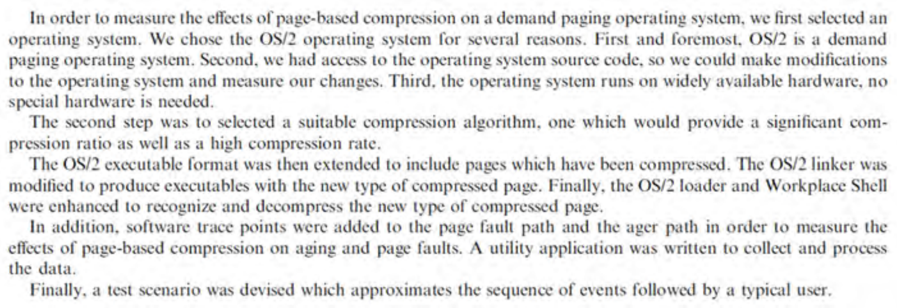
As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

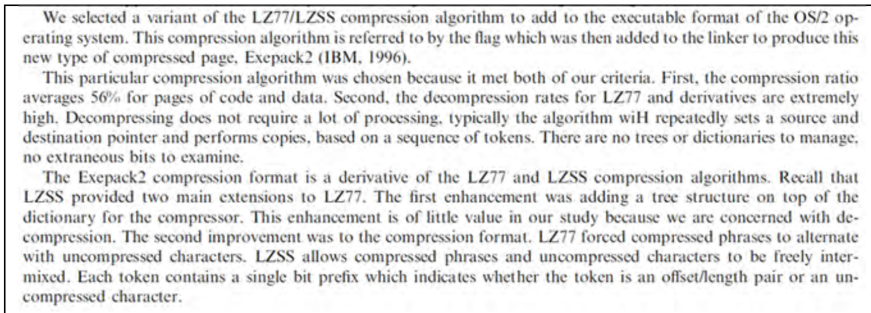
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

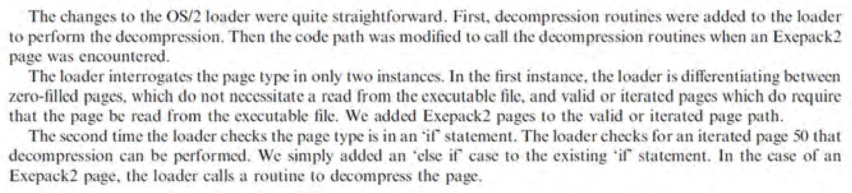
Wynn Article, 3.1


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

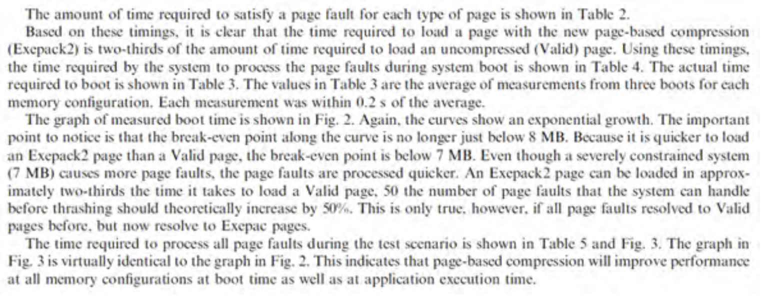
Wynn Article, 3.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4


The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

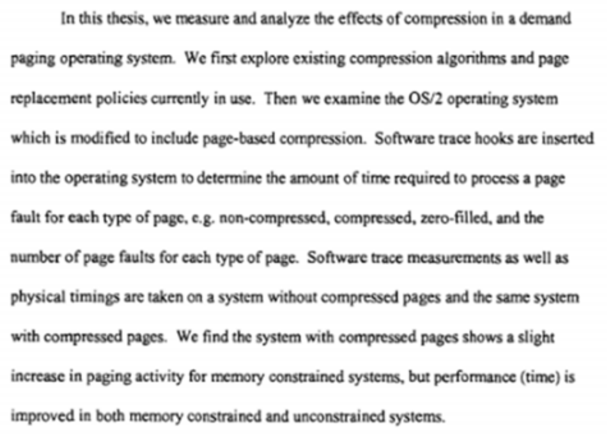
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

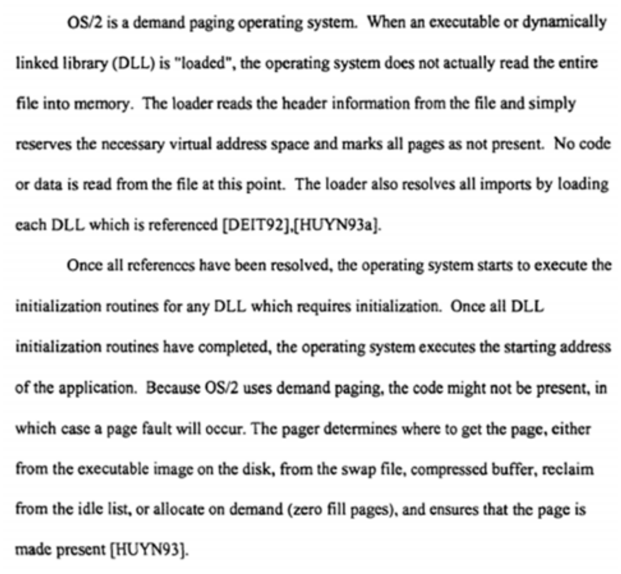
The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2


In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

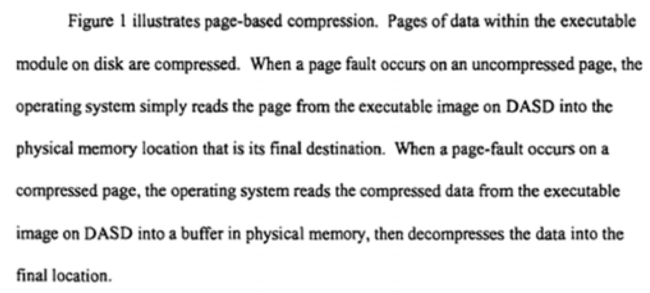
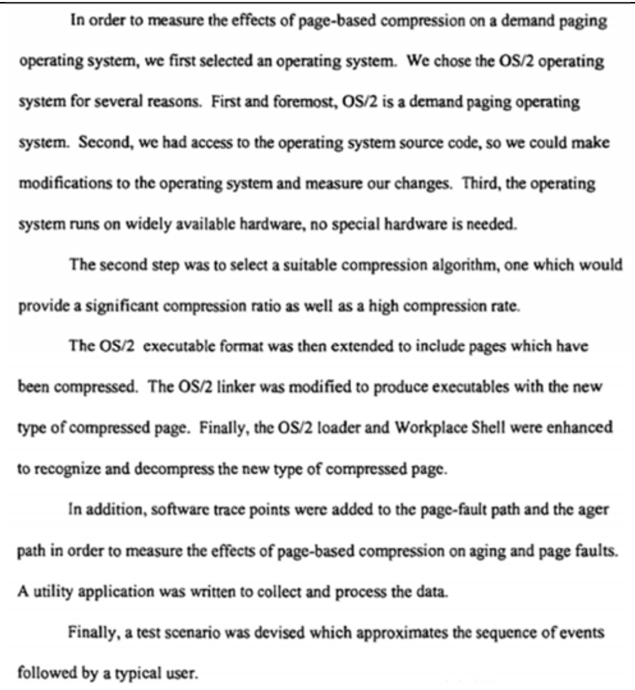


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

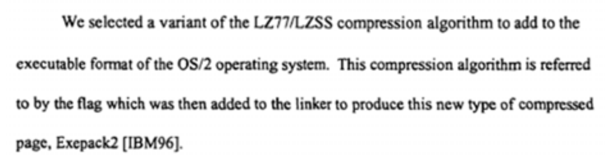
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

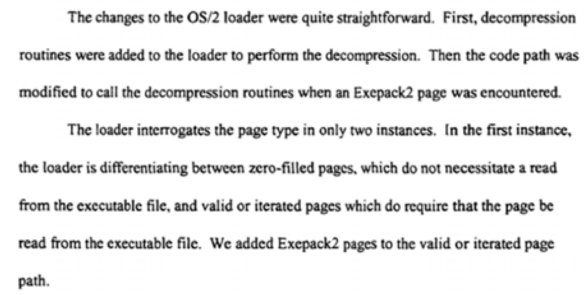
Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1



We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

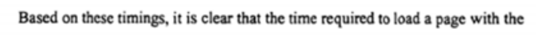
Wynn Thesis, 5.2



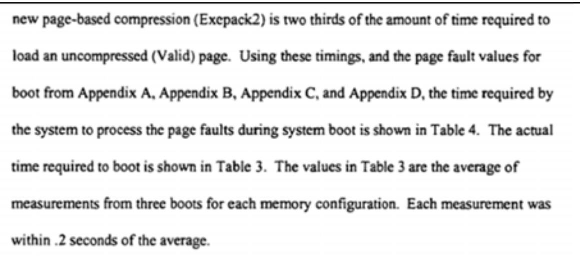
The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

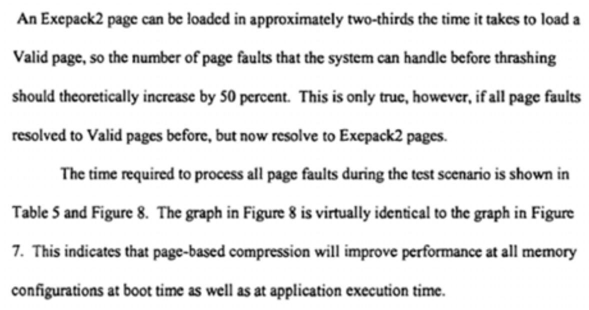


Based on these timings, it is clear that the time required to load a page with the



new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

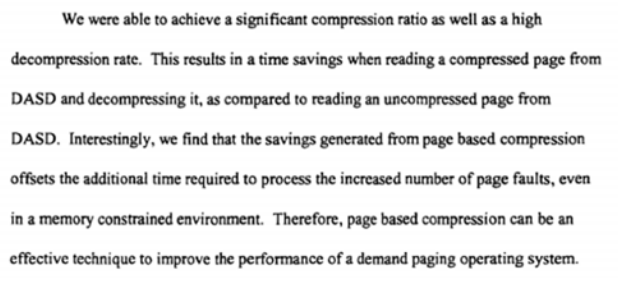
Wynn Thesis, 6.3



An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3



We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

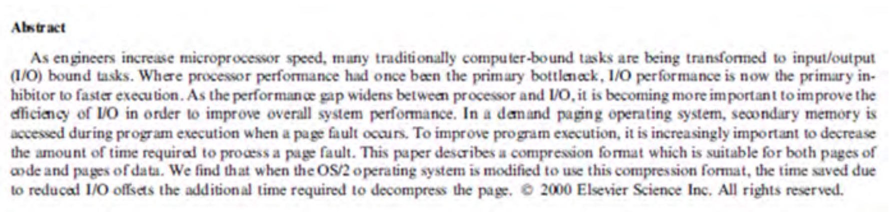
Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

1.3 decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and	Wynn, as evidenced by the example citations below, discloses "decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and"

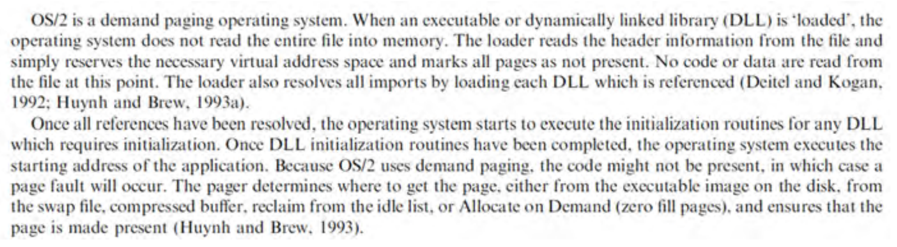
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:


Abstract

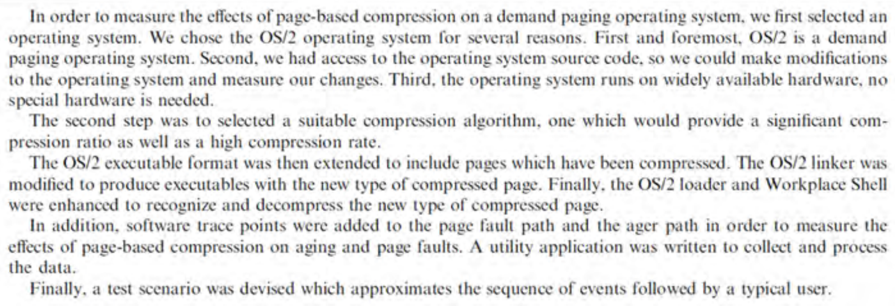
As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

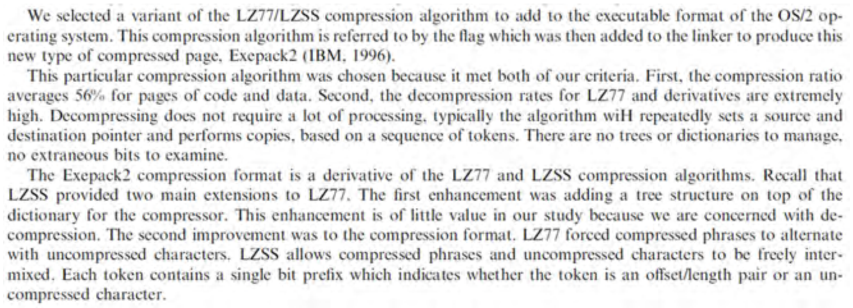
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

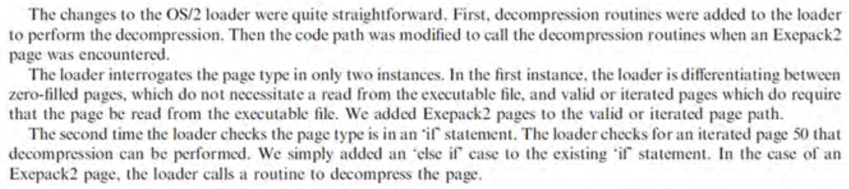
Wynn Article, 3.1


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

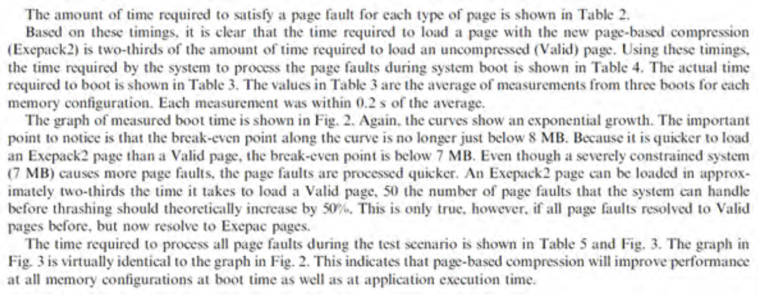
Wynn Article, 3.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4


The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

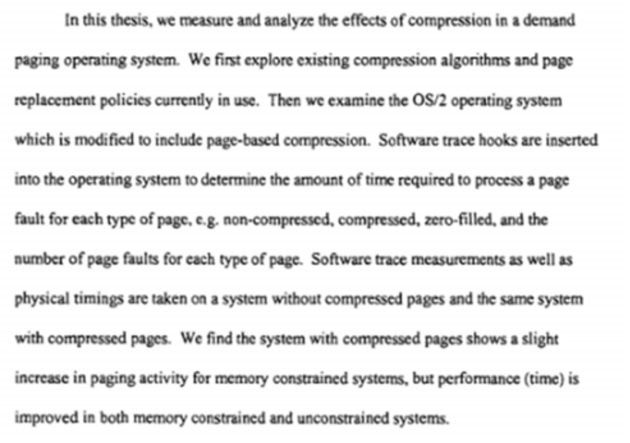
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

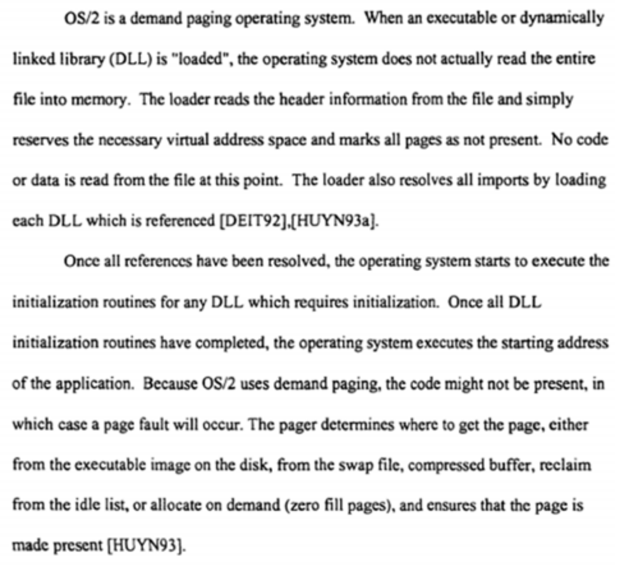
Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

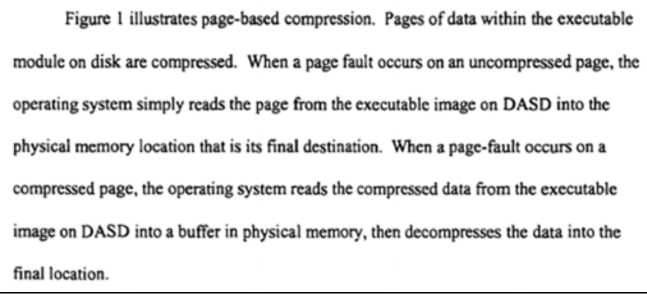
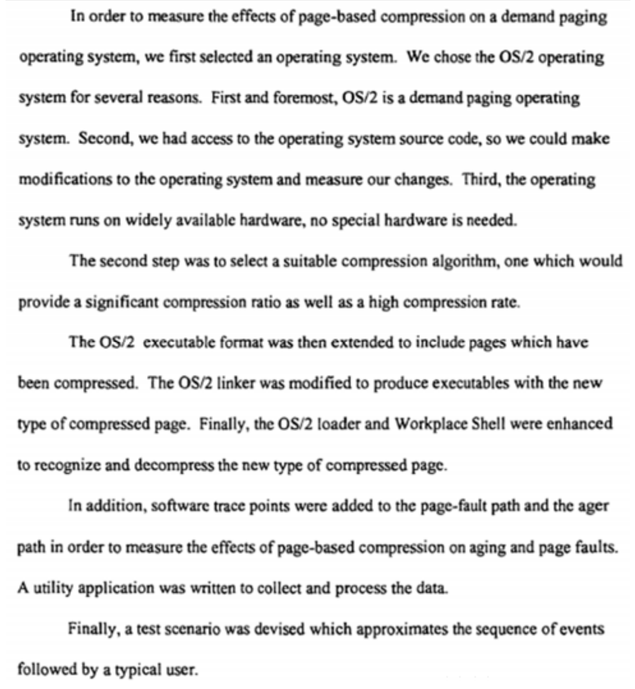


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

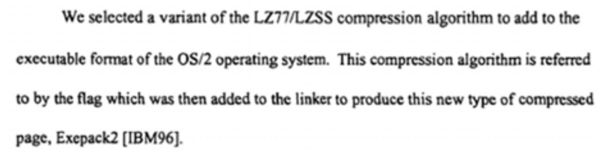
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

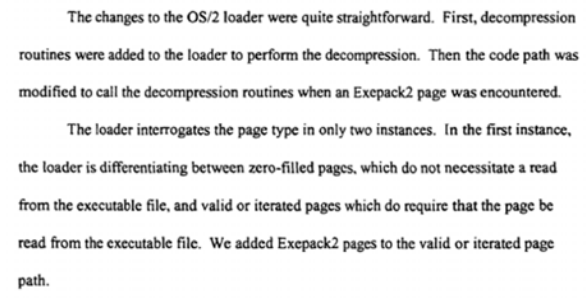
In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

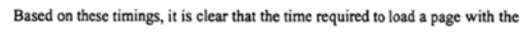

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

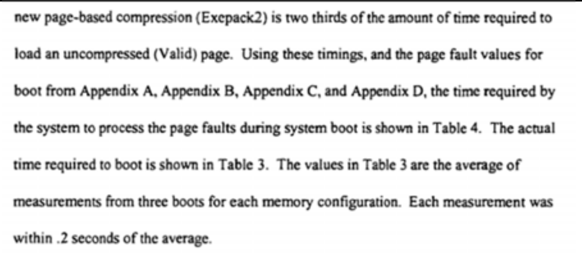
Wynn Thesis, 5.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

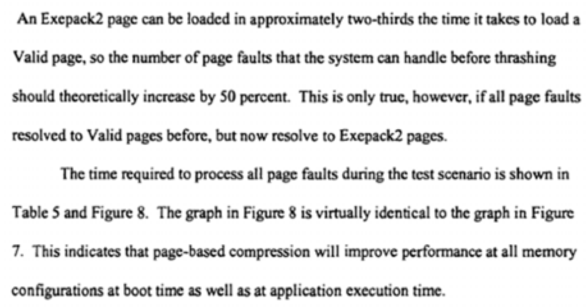
The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5


Based on these timings, it is clear that the time required to load a page with the

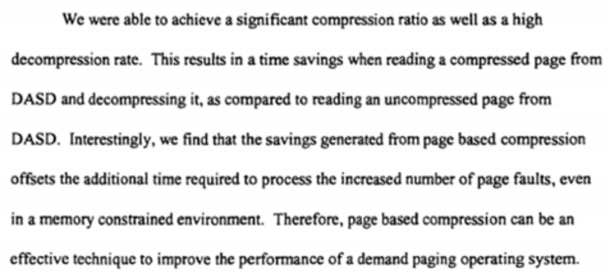

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3


We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

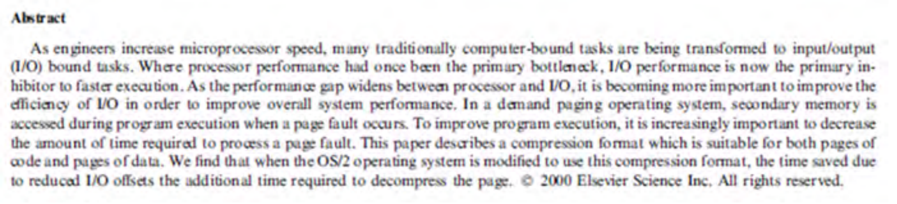
Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

1.4.1 updating the boot data list,	Wynn, as evidenced by the example citations below, discloses "updating the boot data list,"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

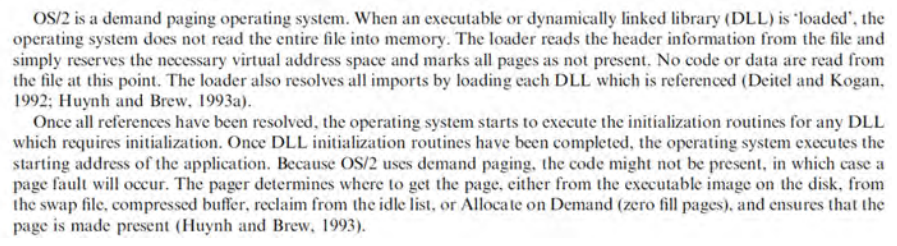
Wynn discloses this limitation:



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

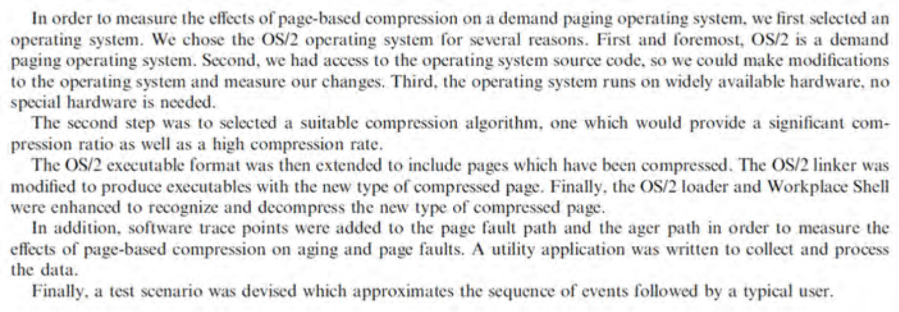
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

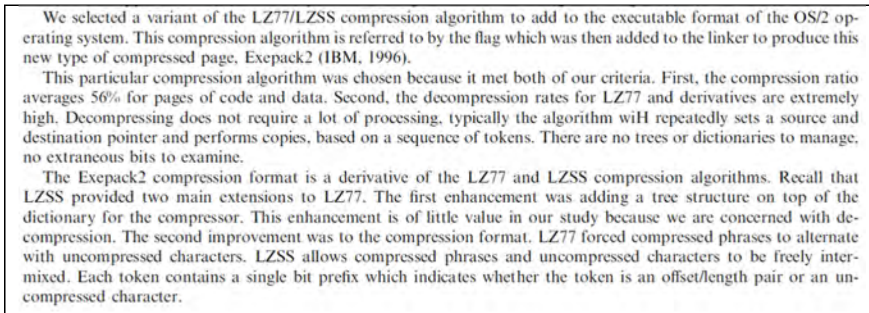
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

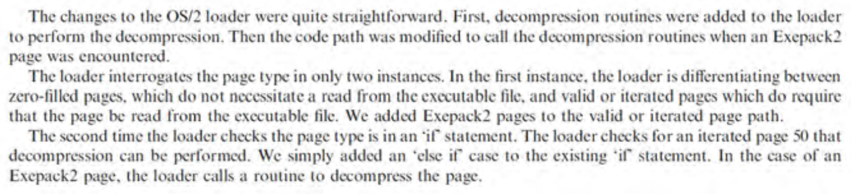


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

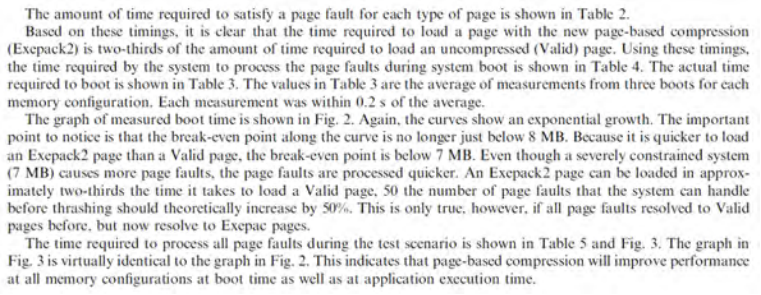


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4



The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

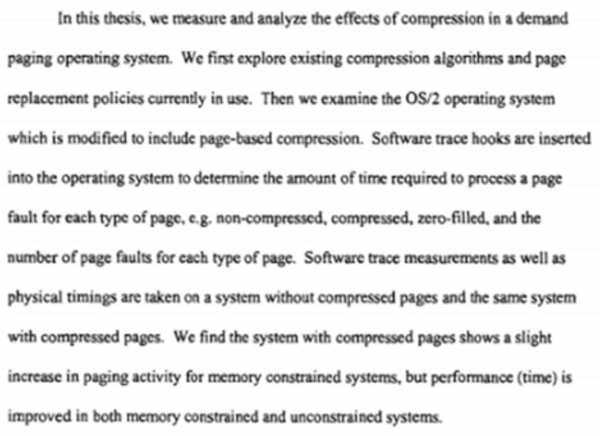
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

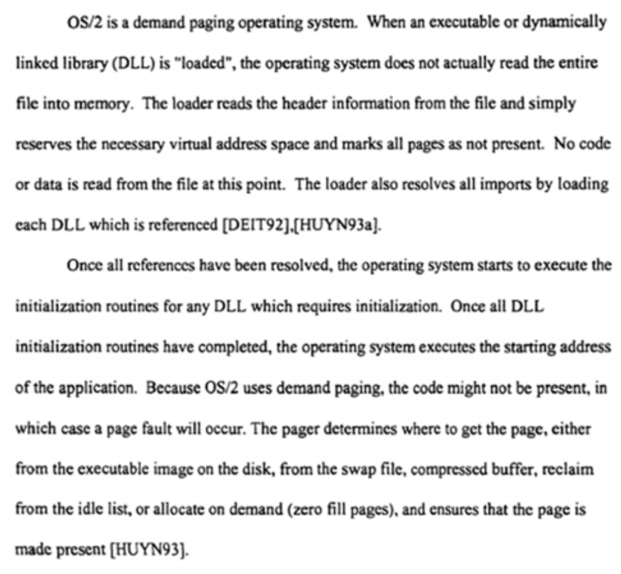
Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

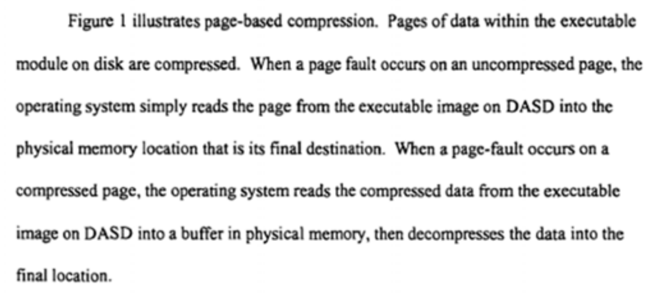
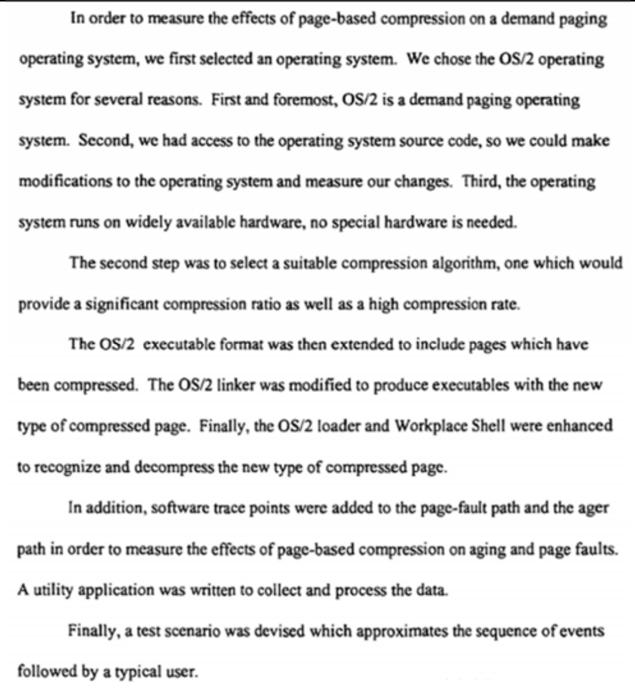


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

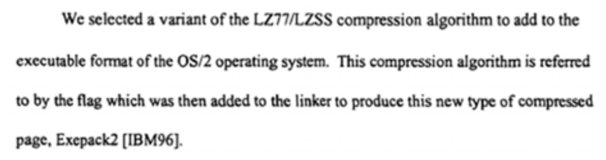
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

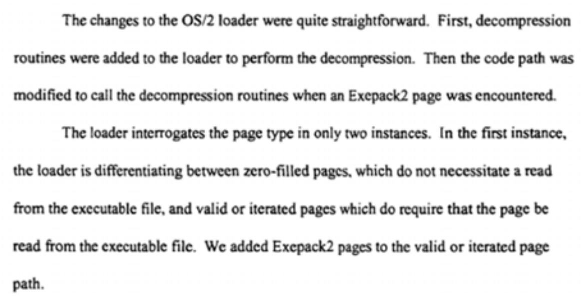
In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

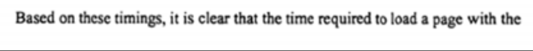

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

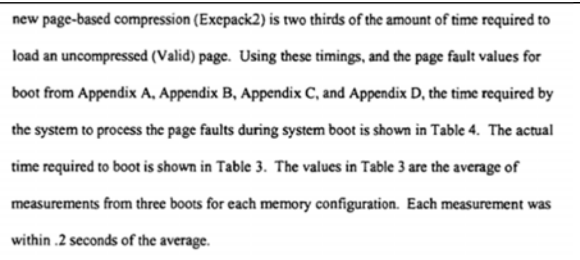
Wynn Thesis, 5.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

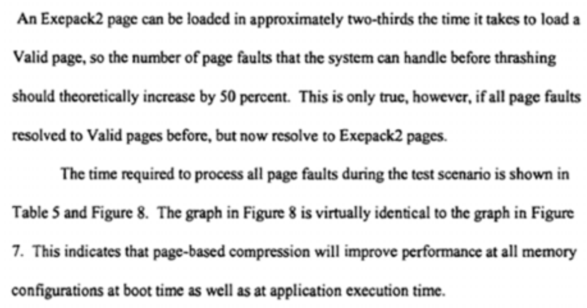
The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5


Based on these timings, it is clear that the time required to load a page with the

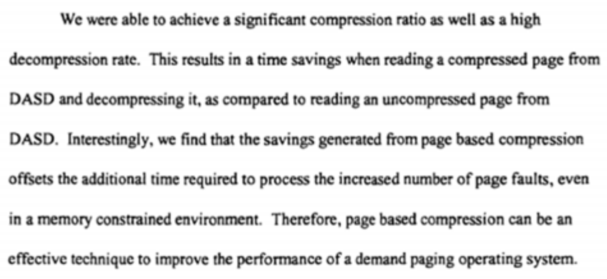

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3


We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

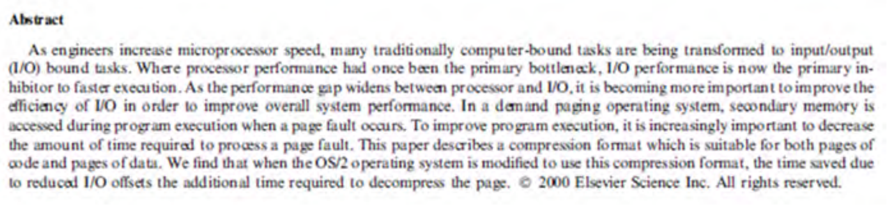
Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

1.4.2 wherein the decompressed portion of boot data comprises a portion of the operating system.	Wynn, as evidenced by the example citations below, discloses "wherein the decompressed portion of boot data comprises a portion of the operating system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the decompressed portion of boot data comprises a portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

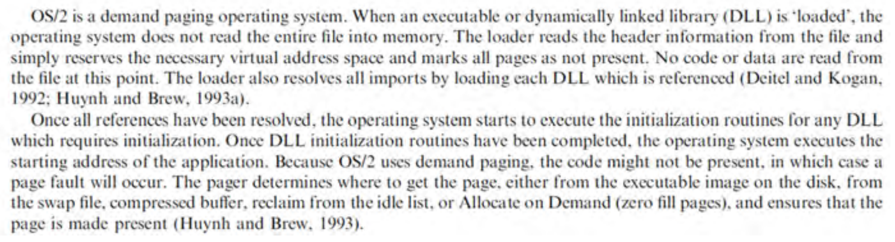
Wynn discloses this limitation:



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

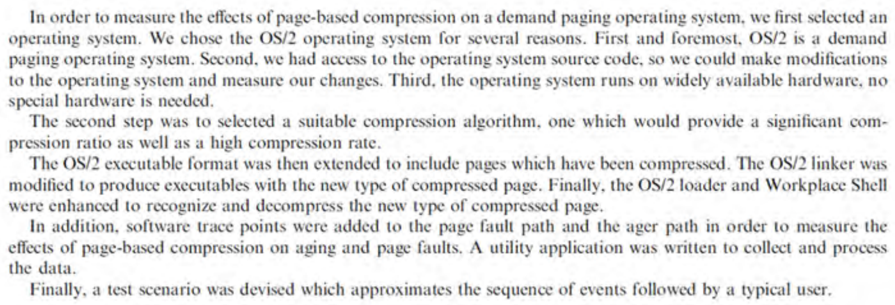
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

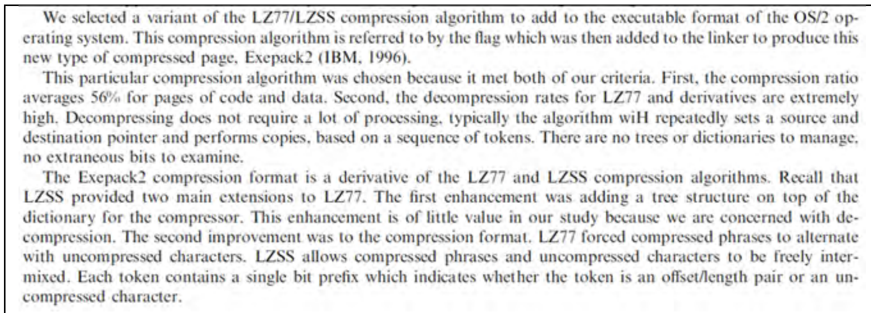
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

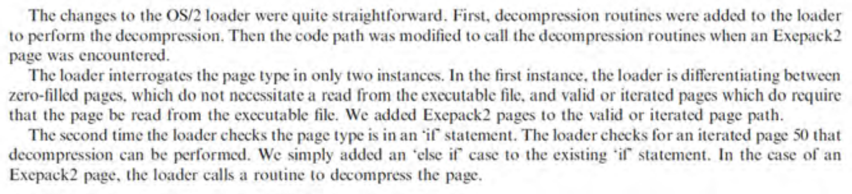
Wynn Article, 3.1


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

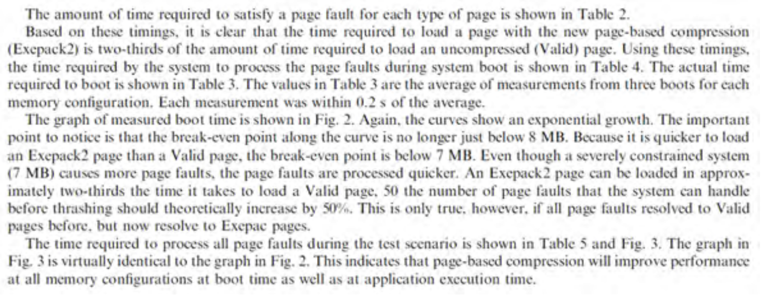
Wynn Article, 3.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4


The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

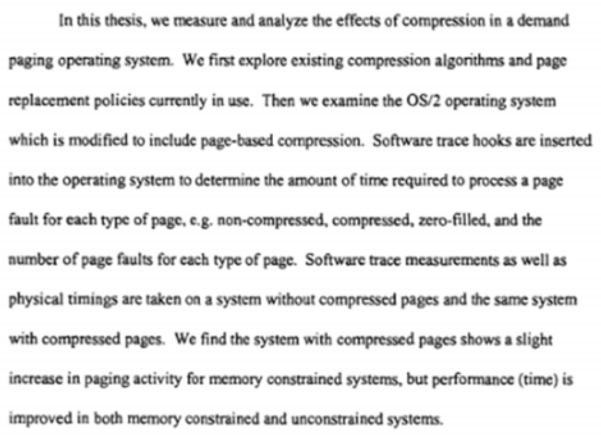
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

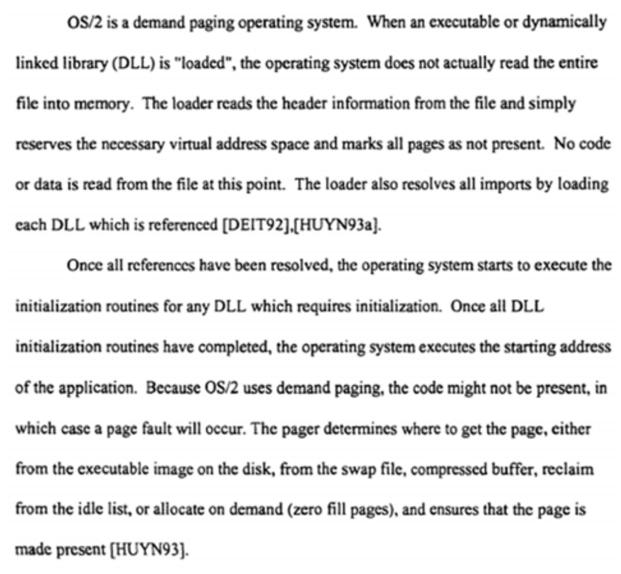
The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2


In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

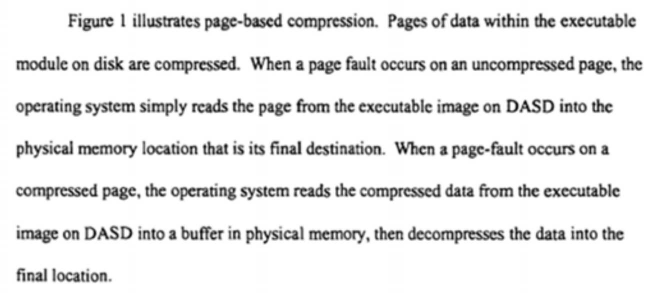
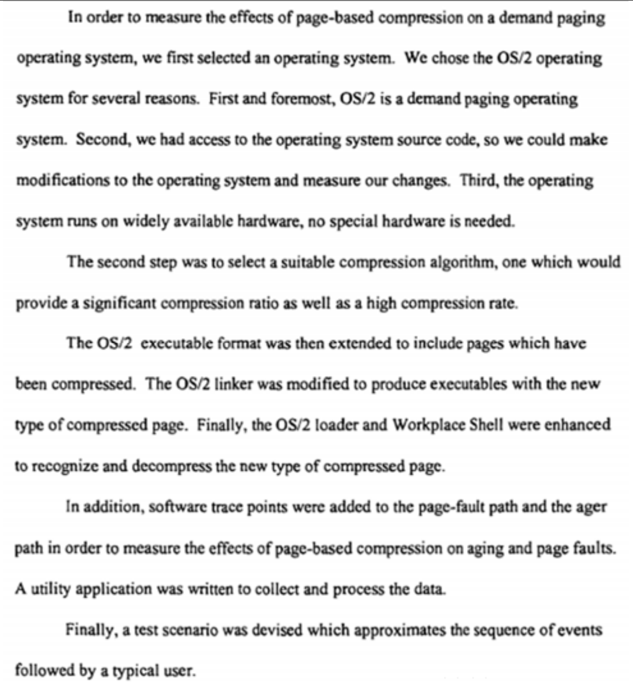


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

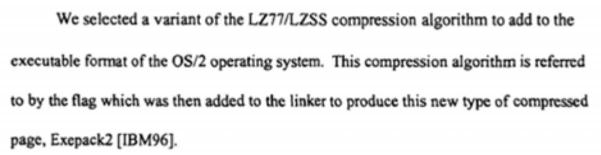
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

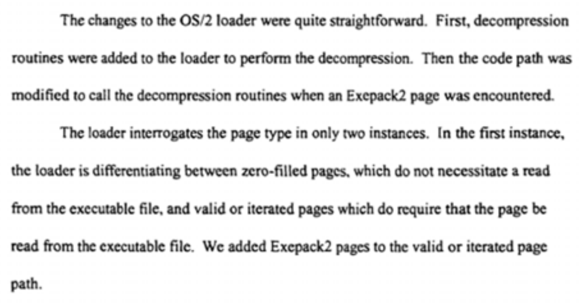
In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

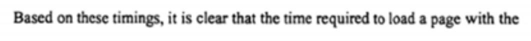

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

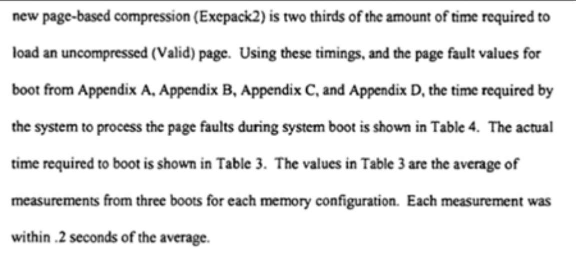
Wynn Thesis, 5.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

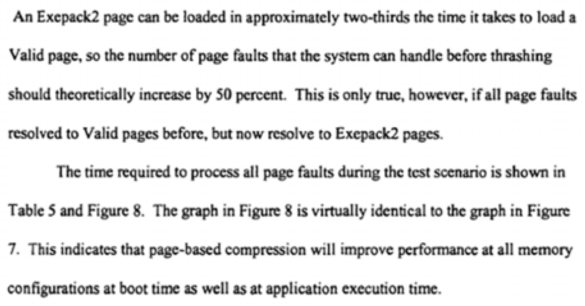
The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5


Based on these timings, it is clear that the time required to load a page with the

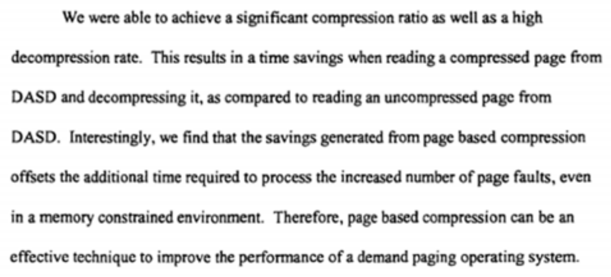

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3


We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

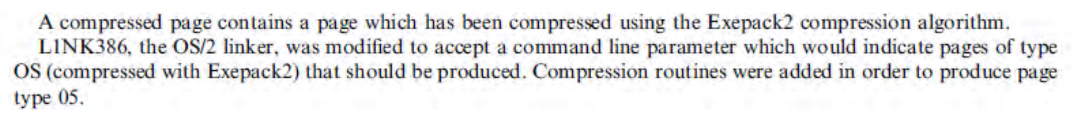
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

2. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list.	Wynn, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claim 1.4.1 above.



A compressed page contains a page which has been compressed using the Exepack2 compression algorithm.
L1NK386, the OS/2 linker, was modified to accept a command line parameter which would indicate pages of type OS (compressed with Exepack2) that should be produced. Compression routines were added in order to produce page type 05.

Wynn Thesis, 3.2

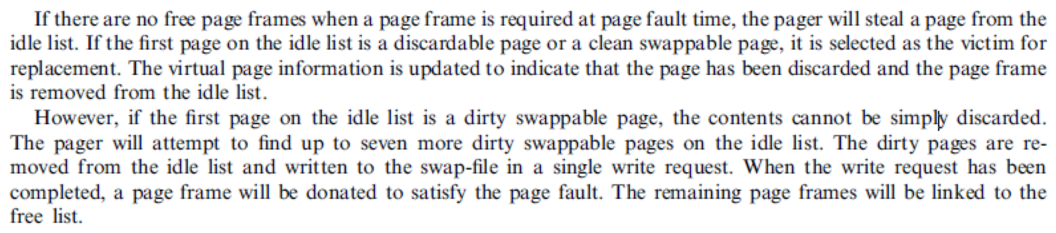
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

3. The method of claim 1, wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list.	Wynn, as evidenced by the example citations below, discloses "wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein removing an association of additional boot data that is associated with the boot data list from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claim 1.4.1 above.



If there are no free page frames when a page frame is required at page fault time, the pager will steal a page from the idle list. If the first page on the idle list is a discardable page or a clean swappable page, it is selected as the victim for replacement. The virtual page information is updated to indicate that the page has been discarded and the page frame is removed from the idle list.

However, if the first page on the idle list is a dirty swappable page, the contents cannot be simply discarded. The pager will attempt to find up to seven more dirty swappable pages on the idle list. The dirty pages are removed from the idle list and written to the swap-file in a single write request. When the write request has been completed, a page frame will be donated to satisfy the page fault. The remaining page frames will be linked to the free list.

Wynn Article, 3.6

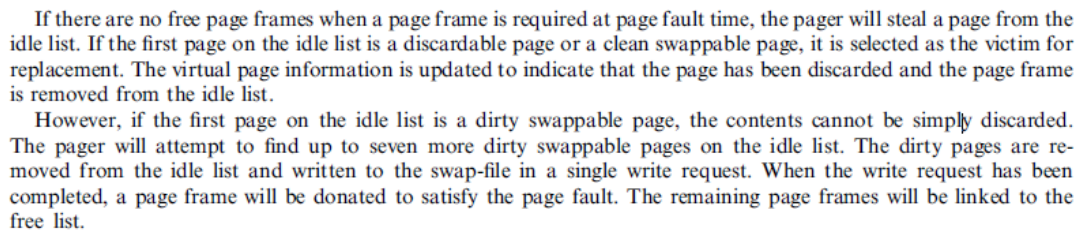
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

4. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data.	Wynn, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list; and compressing a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.4.1, and 2 above.



If there are no free page frames when a page frame is required at page fault time, the pager will steal a page from the idle list. If the first page on the idle list is a discardable page or a clean swappable page, it is selected as the victim for replacement. The virtual page information is updated to indicate that the page has been discarded and the page frame is removed from the idle list.

However, if the first page on the idle list is a dirty swappable page, the contents cannot be simply discarded. The pager will attempt to find up to seven more dirty swappable pages on the idle list. The dirty pages are removed from the idle list and written to the swap-file in a single write request. When the write request has been completed, a page frame will be donated to satisfy the page fault. The remaining page frames will be linked to the free list.

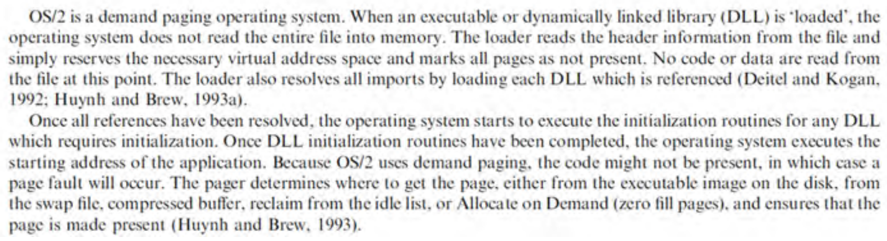
Wynn Article, 3.6

5 (Preamble) A method for booting a computer system, the method comprising:	Wynn, as evidenced by the example citations below, discloses "a method for booting a computer system, the method comprising:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

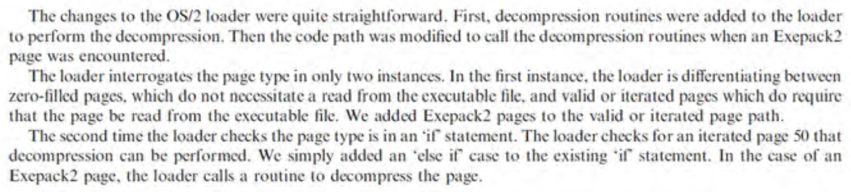
Wynn discloses this limitation:

See Claims 1-1.4.2 above.


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

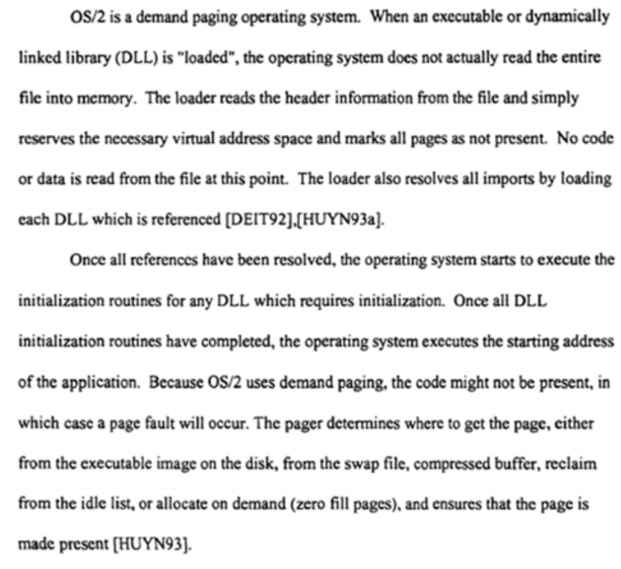
Wynn Article, 2.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

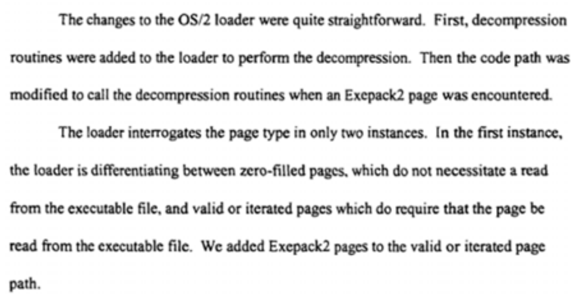
The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page so that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

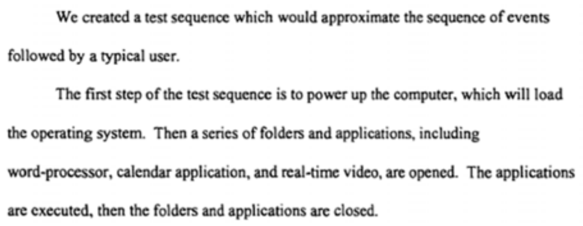
Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

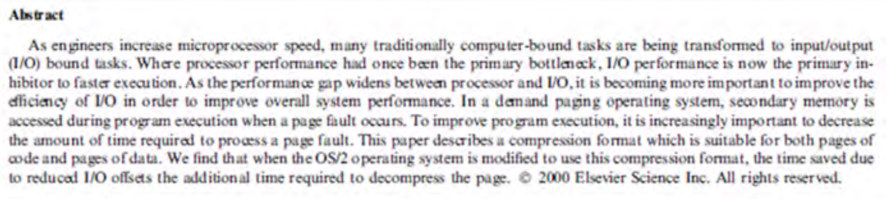

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5


We created a test sequence which would approximate the sequence of events followed by a typical user.

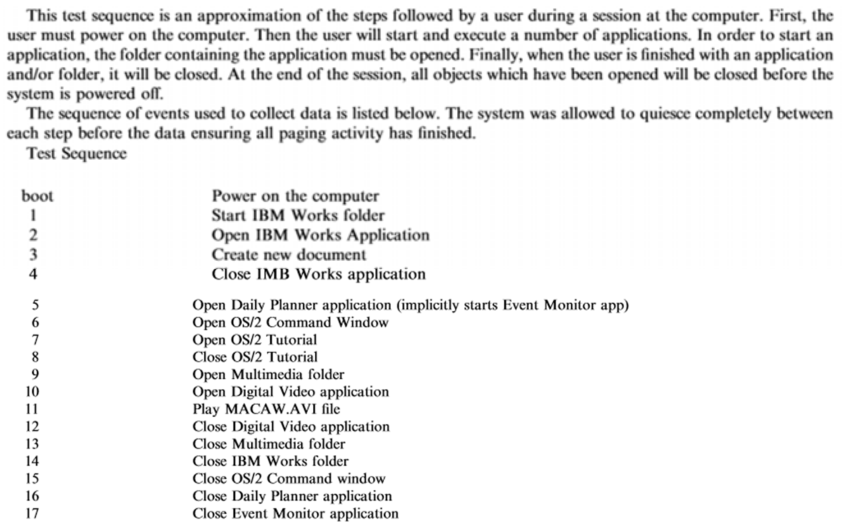
The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract



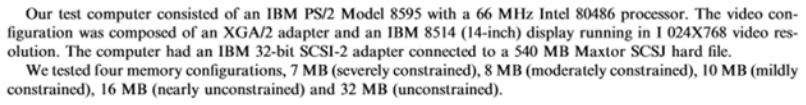
This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

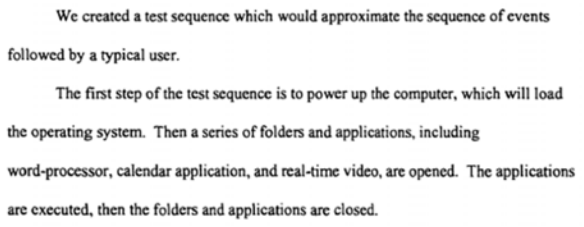
Wynn Article, 3.8



Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1 024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSJ hard file.

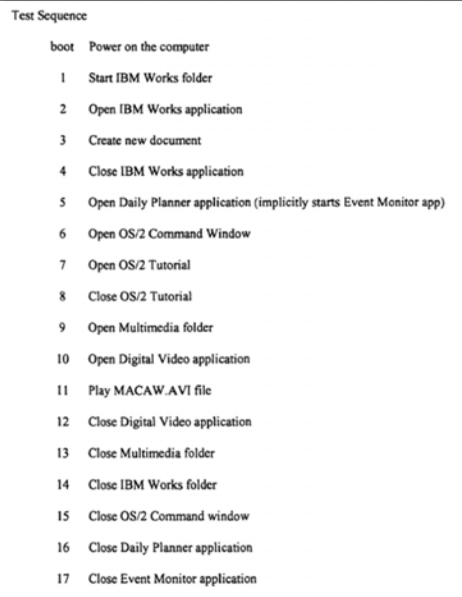
We tested four memory configurations, 7 MB (severely constrained), 8 MB (moderately constrained), 10 MB (mildly constrained), 16 MB (nearly unconstrained) and 32 MB (unconstrained).

Wynn Article, 3.9



We created a test sequence which would approximate the sequence of events followed by a typical user.

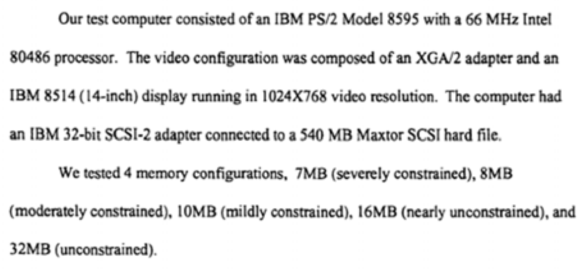
The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.



Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Thesis, 5.9



Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSI hard file.

We tested 4 memory configurations, 7MB (severely constrained), 8MB (moderately constrained), 10MB (mildly constrained), 16MB (nearly unconstrained), and 32MB (unconstrained).

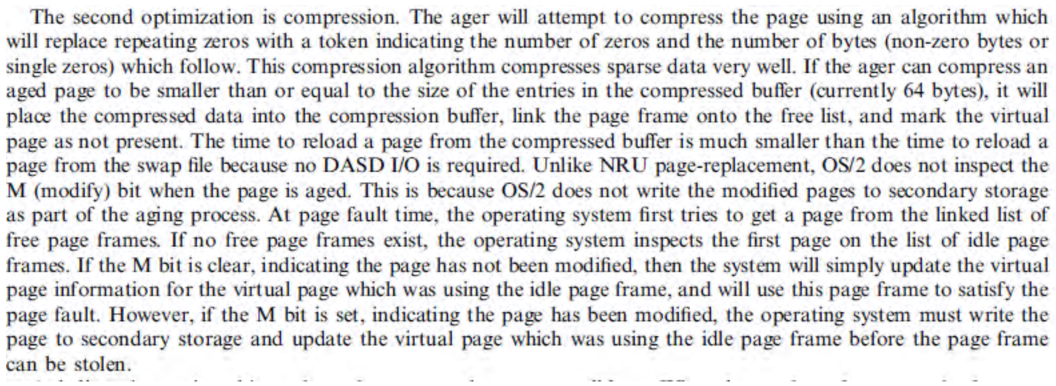
Wynn Thesis, 5.10

5.1 storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory;	Wynn, as evidenced by the example citations below, discloses "storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claim 1.1 above.


The second optimization is compression. The ager will attempt to compress the page using an algorithm which will replace repeating zeros with a token indicating the number of zeros and the number of bytes (non-zero bytes or single zeros) which follow. This compression algorithm compresses sparse data very well. If the ager can compress an aged page to be smaller than or equal to the size of the entries in the compressed buffer (currently 64 bytes), it will place the compressed data into the compression buffer, link the page frame onto the free list, and mark the virtual page as not present. The time to reload a page from the compressed buffer is much smaller than the time to reload a page from the swap file because no DASD I/O is required. Unlike NRU page-replacement, OS/2 does not inspect the M (modify) bit when the page is aged. This is because OS/2 does not write the modified pages to secondary storage as part of the aging process. At page fault time, the operating system first tries to get a page from the linked list of free page frames. If no free page frames exist, the operating system inspects the first page on the list of idle page frames. If the M bit is clear, indicating the page has not been modified, then the system will simply update the virtual page information for the virtual page which was using the idle page frame, and will use this page frame to satisfy the page fault. However, if the M bit is set, indicating the page has been modified, the operating system must write the page to secondary storage and update the virtual page which was using the idle page frame before the page frame can be stolen.

Wynn Article, 2.2

5.2 loading the stored compressed boot data from the first memory;	Wynn, as evidenced by the example citations below, discloses “loading the stored compressed boot data from the first memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the stored compressed boot data from the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 above.	

5.3 accessing the loaded compressed boot data;	Wynn, as evidenced by the example citations below, discloses "accessing the loaded compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.2 above.	

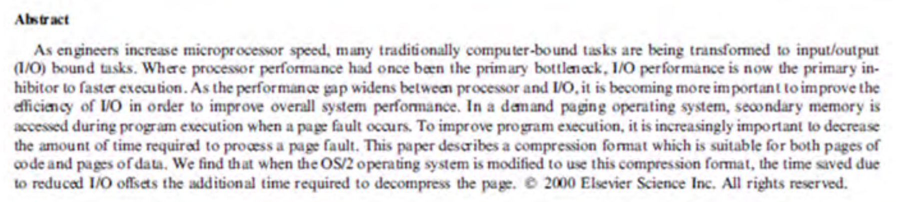
5.4 decompressing the accessed compressed boot data;	Wynn, as evidenced by the example citations below, discloses "decompressing the accessed compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.3 above.	

5.5 utilizing the decompressed boot data to at least partially boot the computer system;	Wynn, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to at least partially boot the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

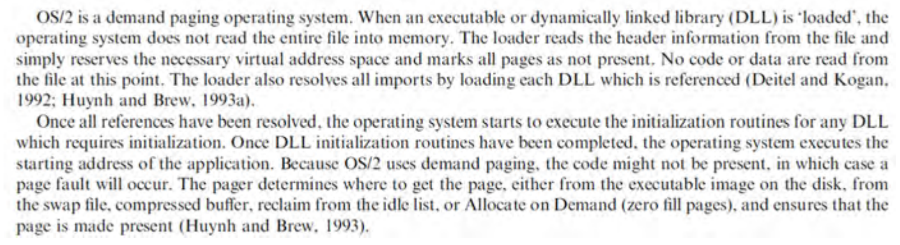
See Claims 1.3 and 1.4.2 above.



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

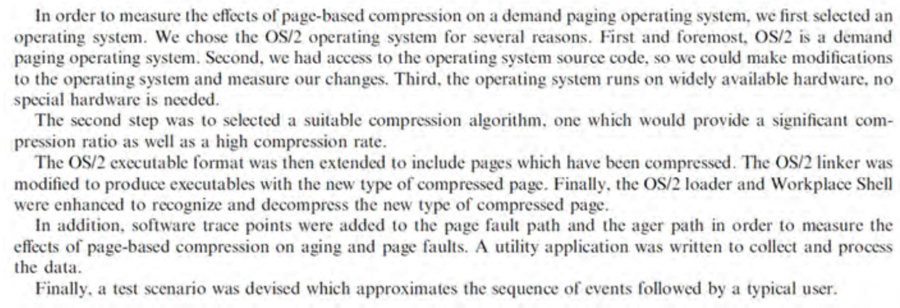
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

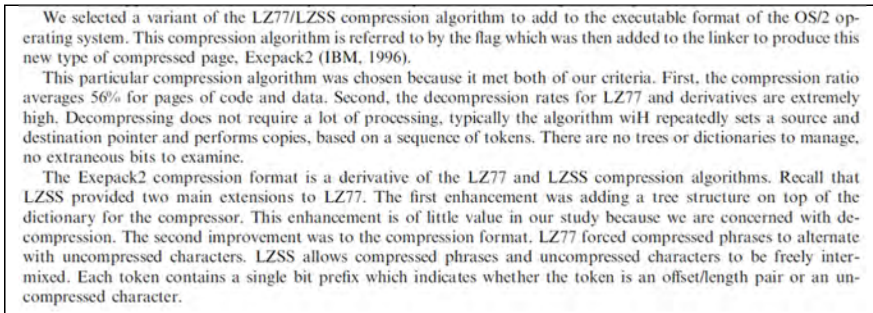
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

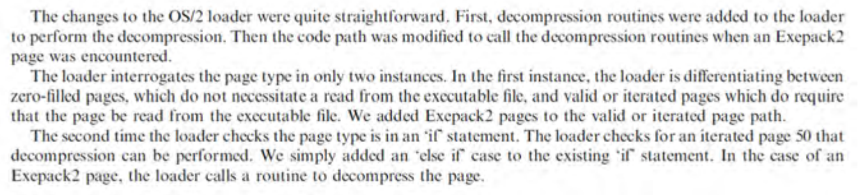


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

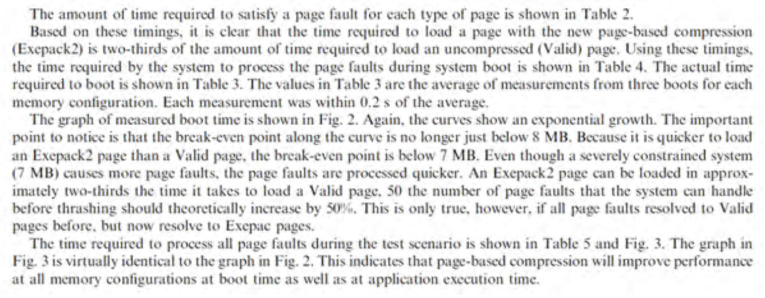


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4



The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

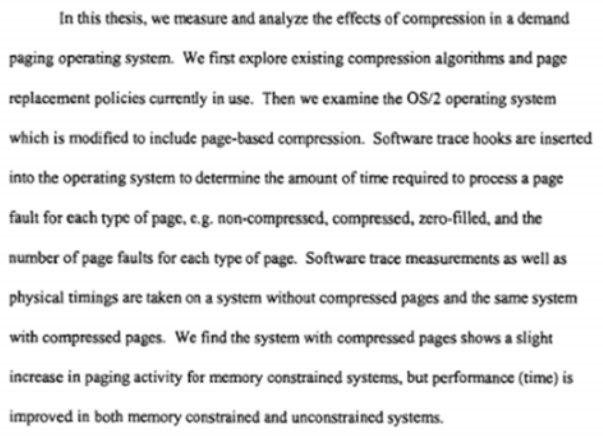
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

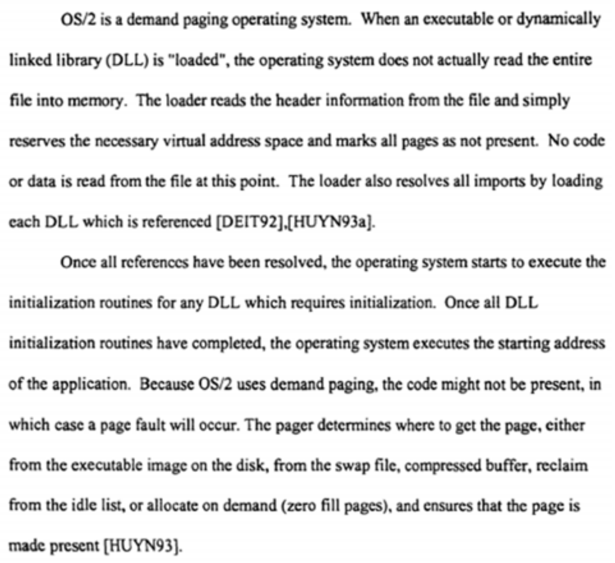
See also, Wynn Article, Table 3 and Fig. 2



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

Wynn Thesis, Abstract


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

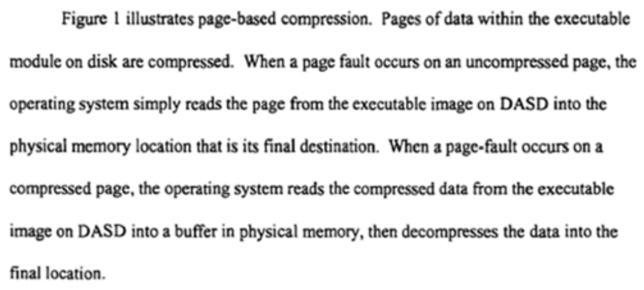
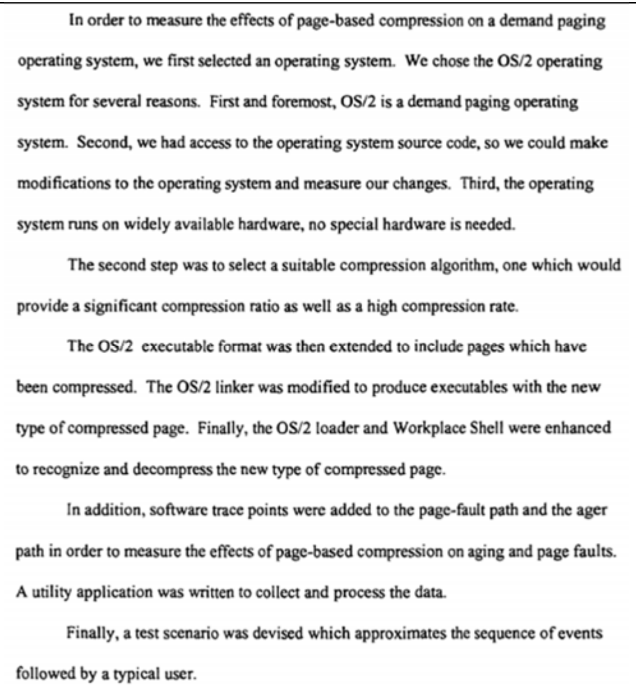


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

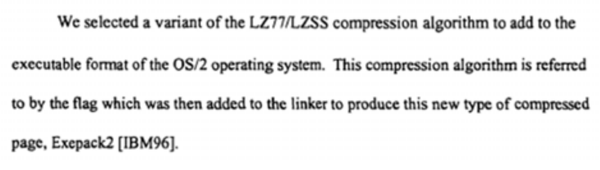
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

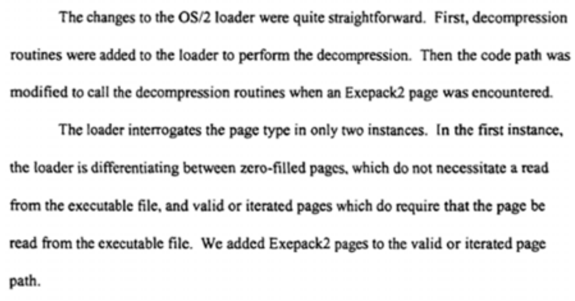
In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

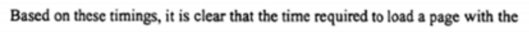

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

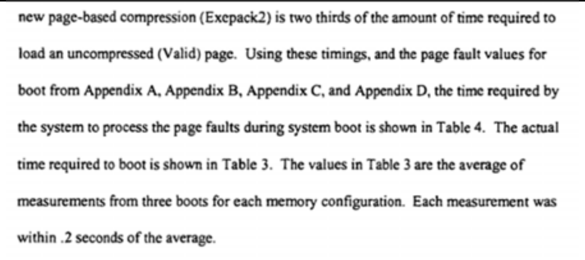
Wynn Thesis, 5.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

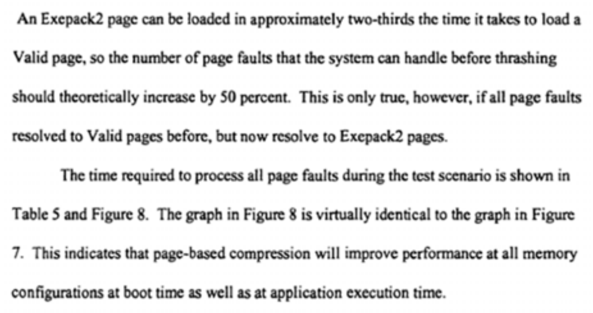
The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5


Based on these timings, it is clear that the time required to load a page with the

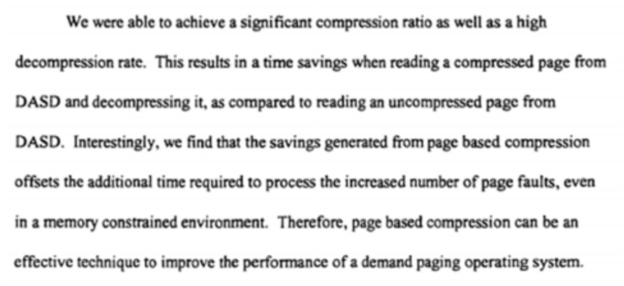

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3


We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

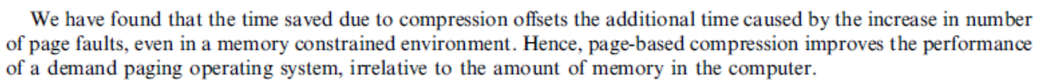
5.6 updating the boot data list;	Wynn, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4.1 above.	

5.7 wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.	Wynn, as evidenced by the example citations below, discloses "wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form"

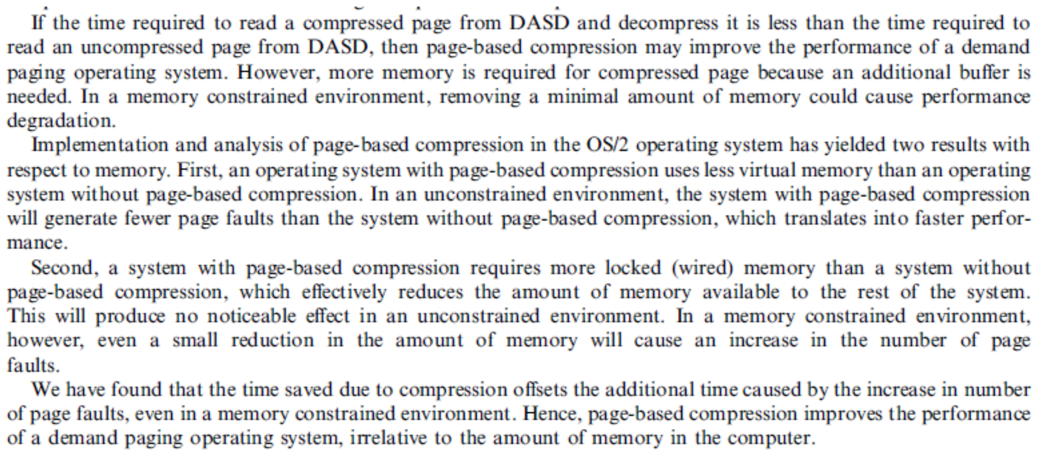
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1-1.3 above.


We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 4.3

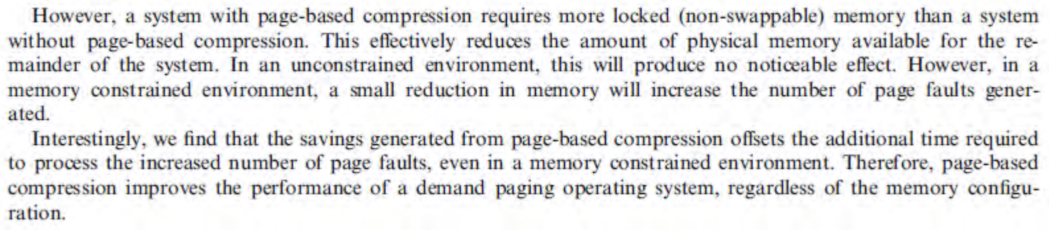

If the time required to read a compressed page from DASD and decompress it is less than the time required to read an uncompressed page from DASD, then page-based compression may improve the performance of a demand paging operating system. However, more memory is required for compressed page because an additional buffer is needed. In a memory constrained environment, removing a minimal amount of memory could cause performance degradation.

Implementation and analysis of page-based compression in the OS/2 operating system has yielded two results with respect to memory. First, an operating system with page-based compression uses less virtual memory than an operating system without page-based compression. In an unconstrained environment, the system with page-based compression will generate fewer page faults than the system without page-based compression, which translates into faster performance.

Second, a system with page-based compression requires more locked (wired) memory than a system without page-based compression, which effectively reduces the amount of memory available to the rest of the system. This will produce no noticeable effect in an unconstrained environment. In a memory constrained environment, however, even a small reduction in the amount of memory will cause an increase in the number of page faults.

We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

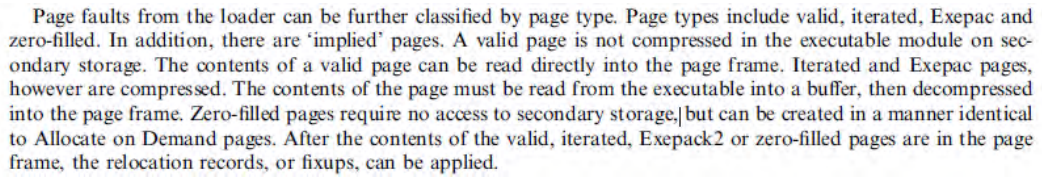
Wynn Article, 1.2



However, a system with page-based compression requires more locked (non-swappable) memory than a system without page-based compression. This effectively reduces the amount of physical memory available for the remainder of the system. In an unconstrained environment, this will produce no noticeable effect. However, in a memory constrained environment, a small reduction in memory will increase the number of page faults generated.

Interestingly, we find that the savings generated from page-based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page-based compression improves the performance of a demand paging operating system, regardless of the memory configuration.

Wynn Article, 5.2



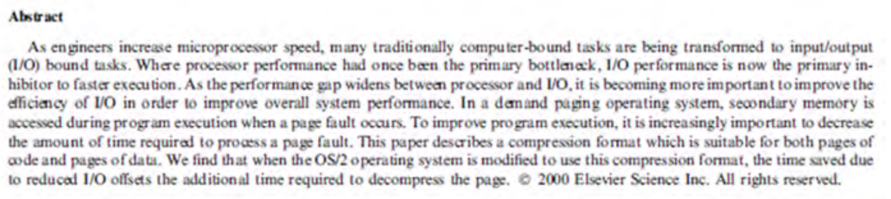
Page faults from the loader can be further classified by page type. Page types include valid, iterated, Exepac and zero-filled. In addition, there are 'implied' pages. A valid page is not compressed in the executable module on secondary storage. The contents of a valid page can be read directly into the page frame. Iterated and Exepac pages, however are compressed. The contents of the page must be read from the executable into a buffer, then decompressed into the page frame. Zero-filled pages require no access to secondary storage, but can be created in a manner identical to Allocate on Demand pages. After the contents of the valid, iterated, Exepack2 or zero-filled pages are in the page frame, the relocation records, or fixups, can be applied.

Wynn Article, 3.6

6 (Preamble) A system comprising: a processor;	Wynn, as evidenced by the example citations below, discloses "a system comprising: a processor:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system comprising: a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

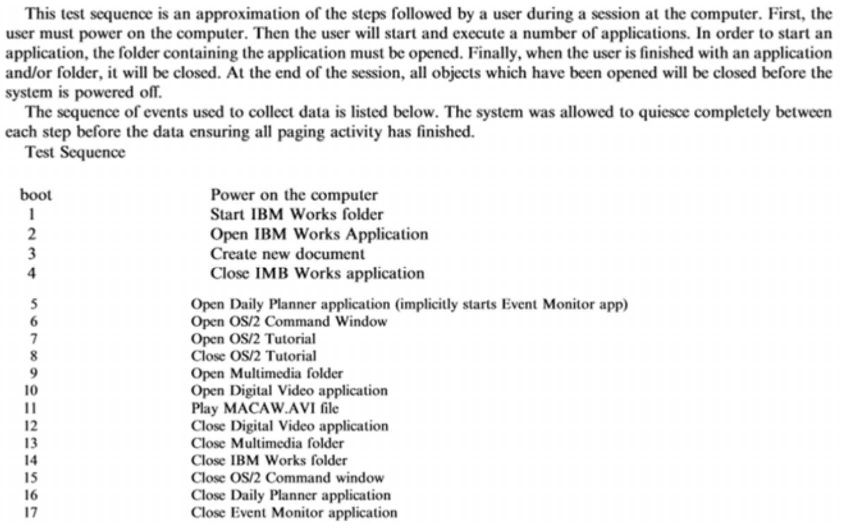
Wynn discloses this limitation:



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract



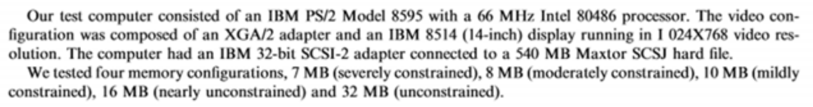
This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

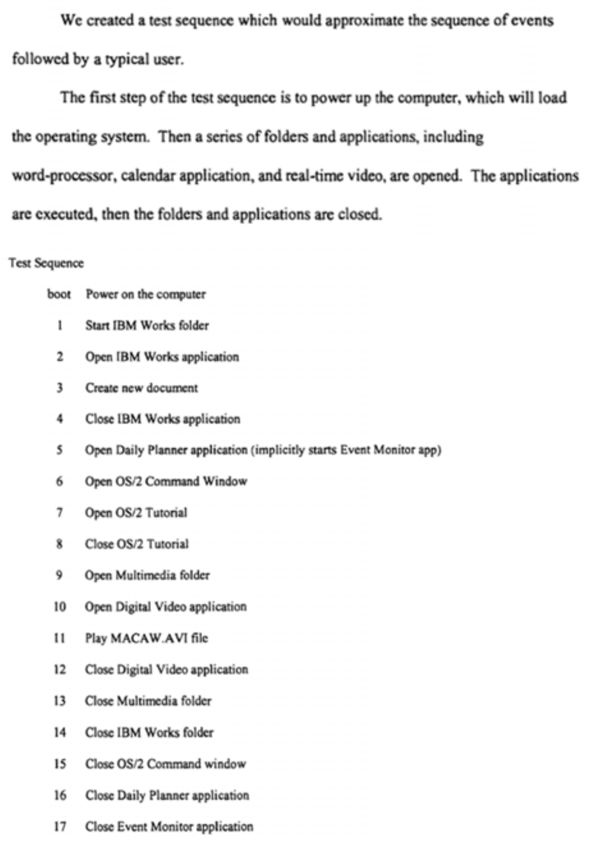
Wynn Article, 3.8



Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1 024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSJ hard file.

We tested four memory configurations, 7 MB (severely constrained), 8 MB (moderately constrained), 10 MB (mildly constrained), 16 MB (nearly unconstrained) and 32 MB (unconstrained).

Wynn Article, 3.9



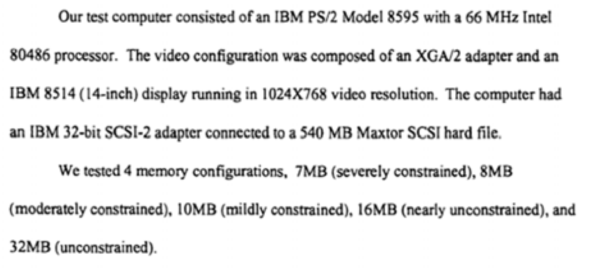
We created a test sequence which would approximate the sequence of events followed by a typical user.

The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Thesis, 5.9



Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSI hard file.

We tested 4 memory configurations, 7MB (severely constrained), 8MB (moderately constrained), 10MB (mildly constrained), 16MB (nearly unconstrained), and 32MB (unconstrained).

Wynn Thesis, 5.10

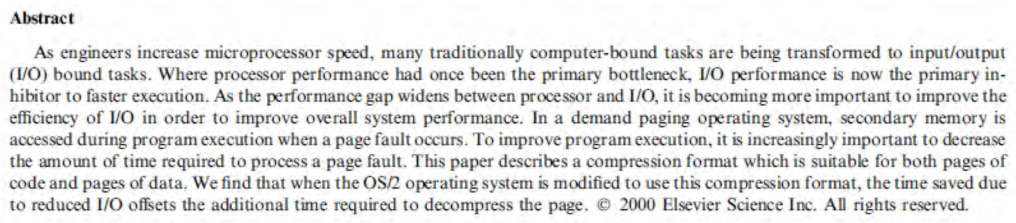
6.1 a memory; and	Wynn, as evidenced by the example citations below, discloses “a memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.2 a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor	Wynn, as evidenced by the example citations below, discloses "a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.1, 1.2, and Claim 6 (Preamble) above.



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

6.3 wherein the processor is configured:	Wynn, as evidenced by the example citations below, discloses “wherein the processor is configured:”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the processor is configured), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Wynn discloses this limitation: *See* Claim 6 (Preamble) above	

6.4 to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory,	Wynn, as evidenced by the example citations below, discloses "to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 above.	

6.5 to access the loaded portion of the boot data in the compressed form,	Wynn, as evidenced by the example citations below, discloses "to access the loaded portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to access the loaded portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.2 above.	

6.6 to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data, and	Wynn, as evidenced by the example citations below, discloses "to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.3 above.	

6.7 to update the boot data list.	Wynn, as evidenced by the example citations below, discloses "to update the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to update the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4.1 above.	

8 (Preamble) A method of loading an operating system for booting a computer system, comprising:	Wynn, as evidenced by the example citations below, discloses "A method of loading an operating system for booting a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method of loading an operating system for booting a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1 (Preamble) above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

8.1 storing a portion of the operating system in a compressed form in a first memory;	Wynn, as evidenced by the example citations below, discloses "storing a portion of the operating system in a compressed form in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing a portion of the operating system in a compressed form in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 6.2 above.	

8.2 loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list;	Wynn, as evidenced by the example citations below, discloses "loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 above.	

8.3 accessing the loaded portion of the operating system from the second memory in the compressed form;	Wynn, as evidenced by the example citations below, discloses "accessing the loaded portion of the operating system from the second memory in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the operating system from the second memory in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.2 above.	

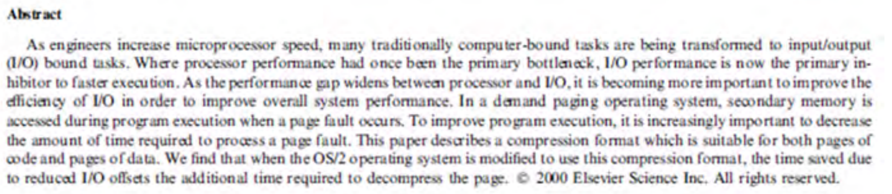
8.4 decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system;	Wynn, as evidenced by the example citations below, discloses "decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.5 utilizing the decompressed portion of the operating system to at least partially boot the computer system; and	Wynn, as evidenced by the example citations below, discloses "utilizing the decompressed portion of the operating system to at least partially boot the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed portion of the operating system to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

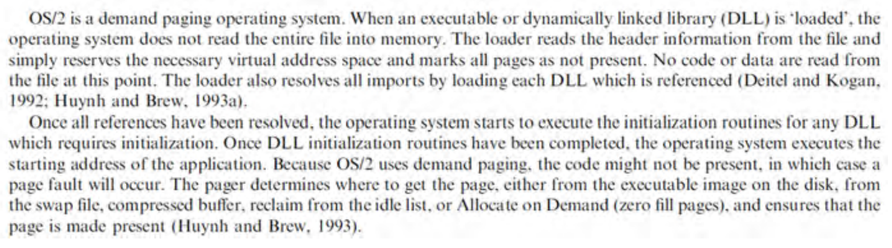
See Claims 1.3 and 1.4.2 above.



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

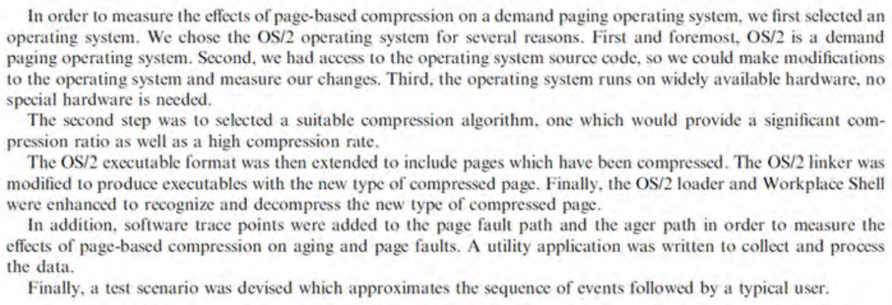
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

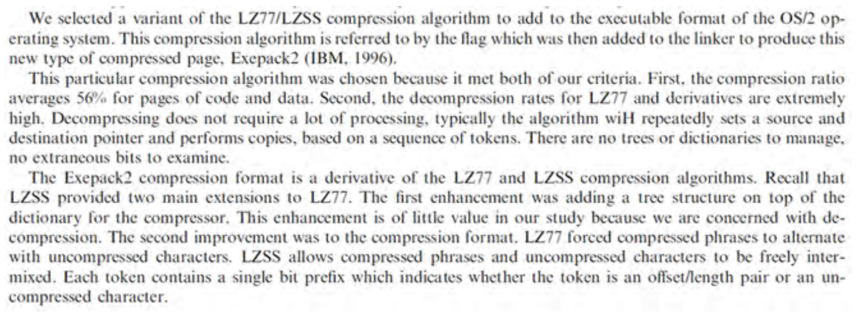
The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

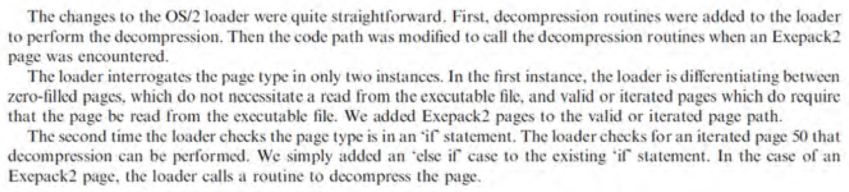
Wynn Article, 3.1


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

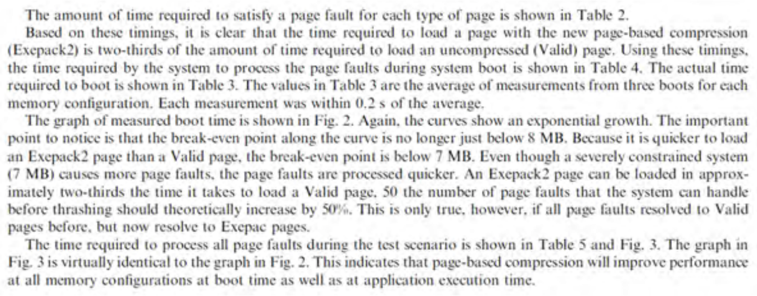
Wynn Article, 3.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an ‘if’ statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an ‘else if’ case to the existing ‘if’ statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4


The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

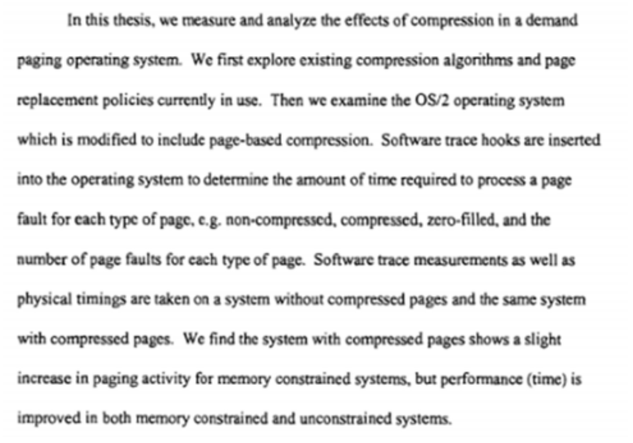
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

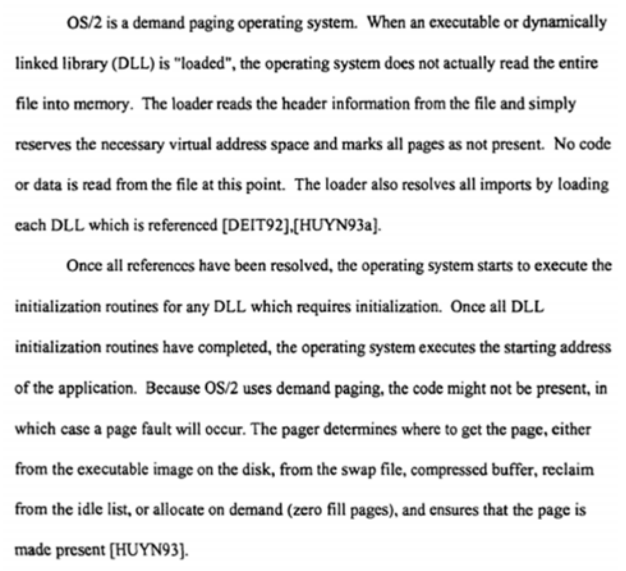
The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2


In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

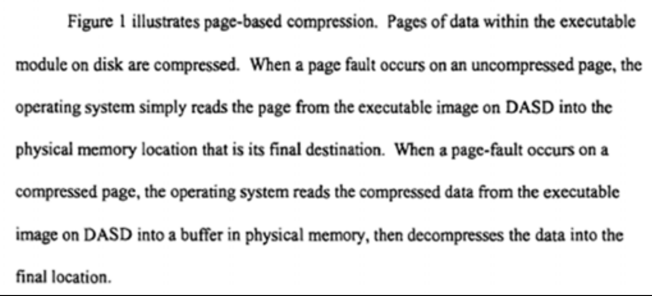
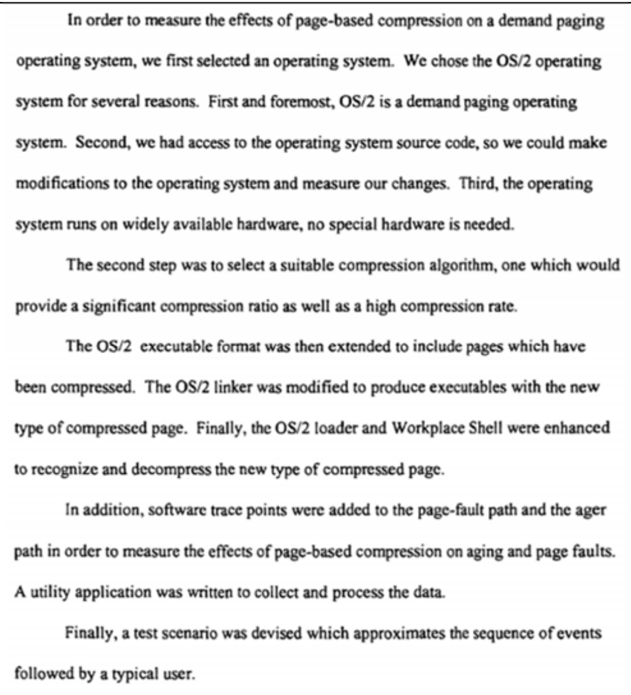


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

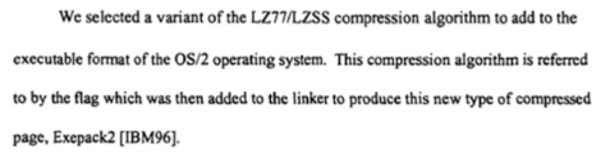
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

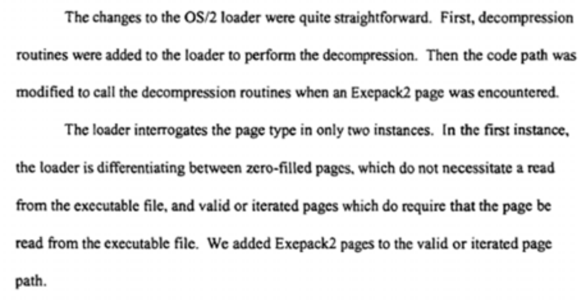
In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1

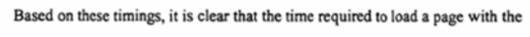

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

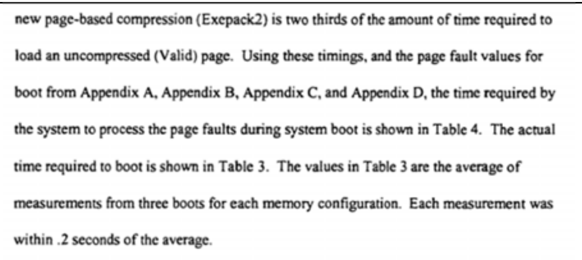
Wynn Thesis, 5.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

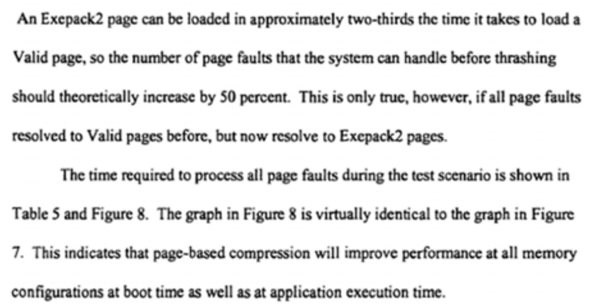
The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5


Based on these timings, it is clear that the time required to load a page with the

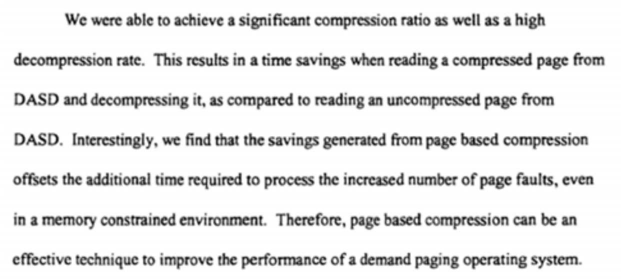

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3


We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

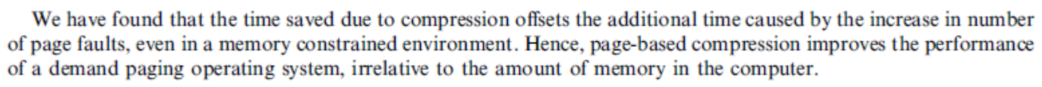
8.6 updating the boot data list,	Wynn, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4.1 above.	

8.7 wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form.	Wynn, as evidenced by the example citations below, discloses "wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form"

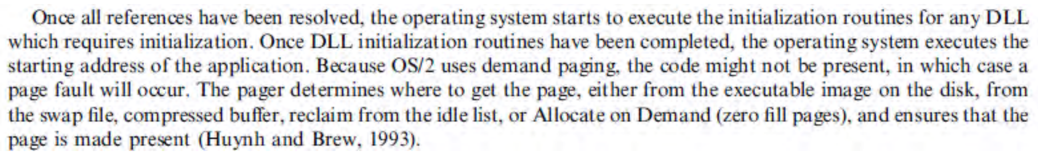
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

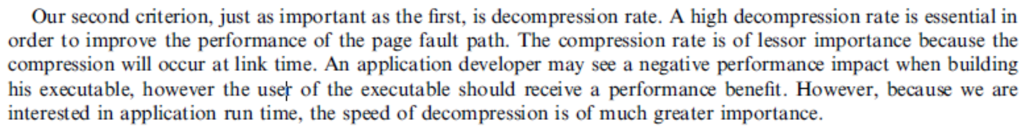
See Claims 1-1.3 and 5.7 above.


We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 4.3


Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2


Our second criterion, just as important as the first, is decompression rate. A high decompression rate is essential in order to improve the performance of the page fault path. The compression rate is of lessor importance because the compression will occur at link time. An application developer may see a negative performance impact when building his executable, however the user of the executable should receive a performance benefit. However, because we are interested in application run time, the speed of decompression is of much greater importance.

Wynn Article, 3.2

If the time required to read a compressed page from DASD and decompress it is less than the time required to read an uncompressed page from DASD, then page-based compression may improve the performance of a demand paging operating system. However, more memory is required for compressed page because an additional buffer is needed. In a memory constrained environment, removing a minimal amount of memory could cause performance degradation.

Implementation and analysis of page-based compression in the OS/2 operating system has yielded two results with respect to memory. First, an operating system with page-based compression uses less virtual memory than an operating system without page-based compression. In an unconstrained environment, the system with page-based compression will generate fewer page faults than the system without page-based compression, which translates into faster performance.

Second, a system with page-based compression requires more locked (wired) memory than a system without page-based compression, which effectively reduces the amount of memory available to the rest of the system. This will produce no noticeable effect in an unconstrained environment. In a memory constrained environment, however, even a small reduction in the amount of memory will cause an increase in the number of page faults.

We have found that the time saved due to compression offsets the additional time caused by the increase in number of page faults, even in a memory constrained environment. Hence, page-based compression improves the performance of a demand paging operating system, irrelative to the amount of memory in the computer.

Wynn Article, 1.2

4.3. Effect of page-based compression on performance

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

However, a system with page-based compression requires more locked (non-swappable) memory than a system without page-based compression. This effectively reduces the amount of physical memory available for the remainder of the system. In an unconstrained environment, this will produce no noticeable effect. However, in a memory constrained environment, a small reduction in memory will increase the number of page faults generated.

Interestingly, we find that the savings generated from page-based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page-based compression improves the performance of a demand paging operating system, regardless of the memory configuration.

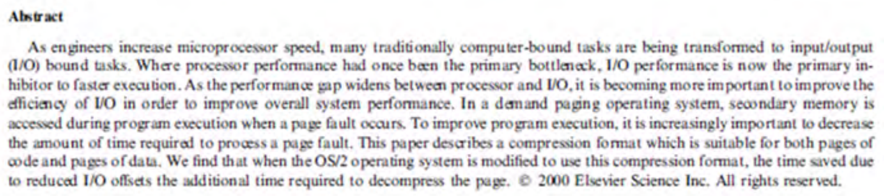
Wynn Article, 5.2

9.1 The method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list; and	Wynn, as evidenced by the example citations below, discloses "the method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, compressing an additional portion of the operating system that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

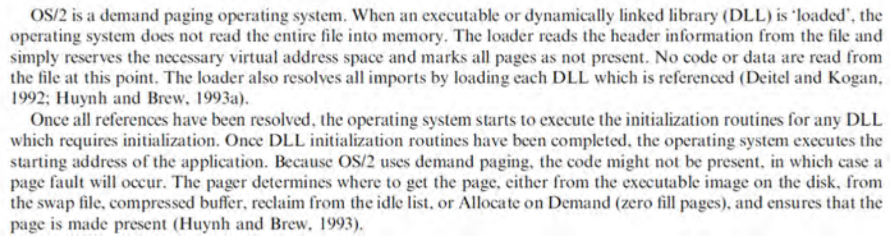
See Claim 1.1 above.



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract

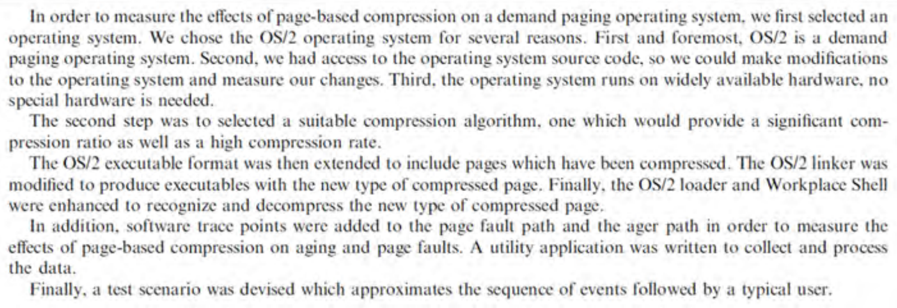


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

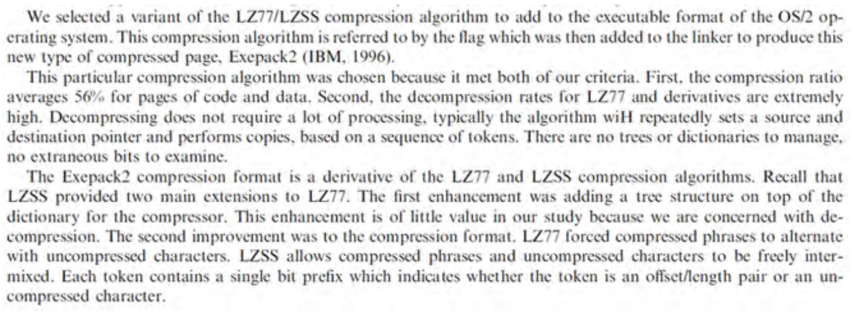
Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

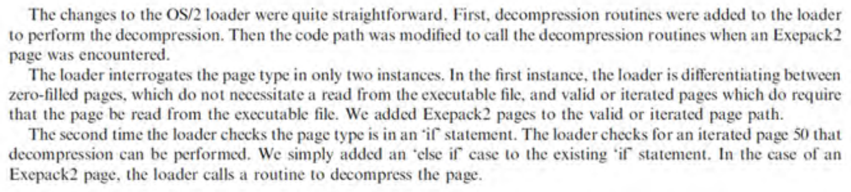
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn


In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.
The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.
In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.
Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

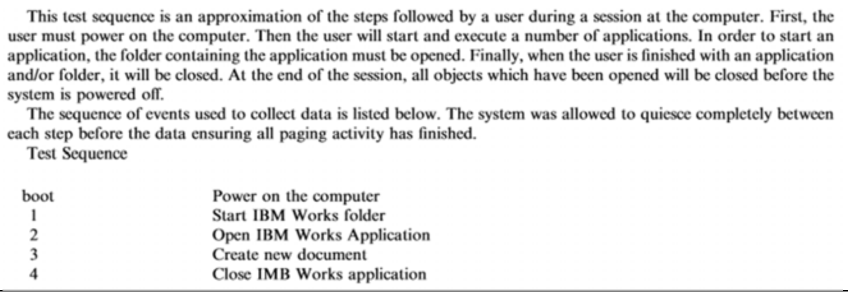
Wynn Article, 3.1


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).
This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.
The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.
The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.
The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

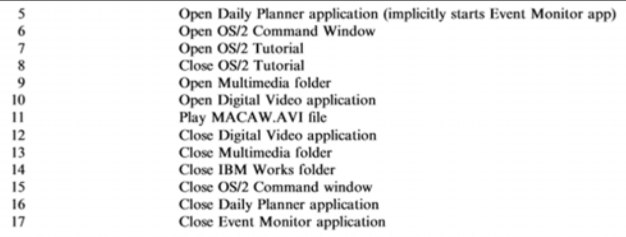
Wynn Article, 3.4


This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.
The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.
Test Sequence

boot	**Power on the computer**
1	**Start IBM Works folder**
2	**Open IBM Works Application**
3	**Create new document**
4	**Close IMB Works application**

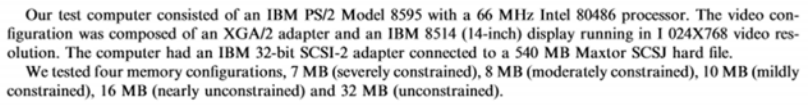
Wynn | Claim 9.1

"The method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list"



5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

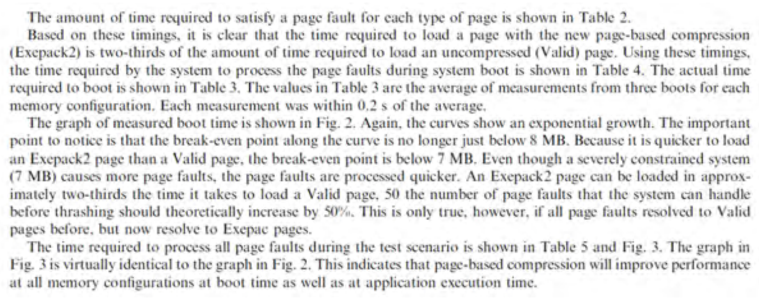
Wynn Article, 3.8



Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in I 024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSJ hard file.

We tested four memory configurations, 7 MB (severely constrained), 8 MB (moderately constrained), 10 MB (mildly constrained), 16 MB (nearly unconstrained) and 32 MB (unconstrained).

Wynn Article, 3.9



The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

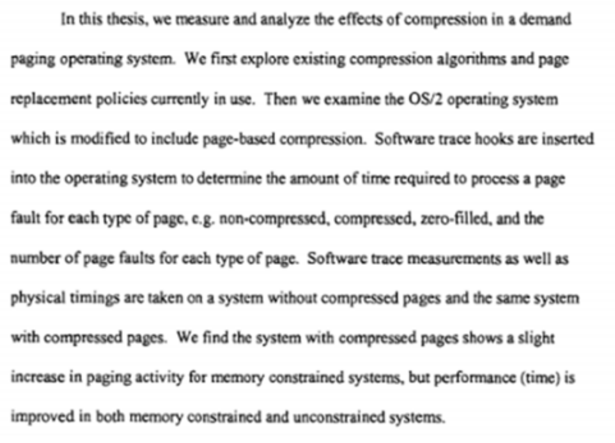
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

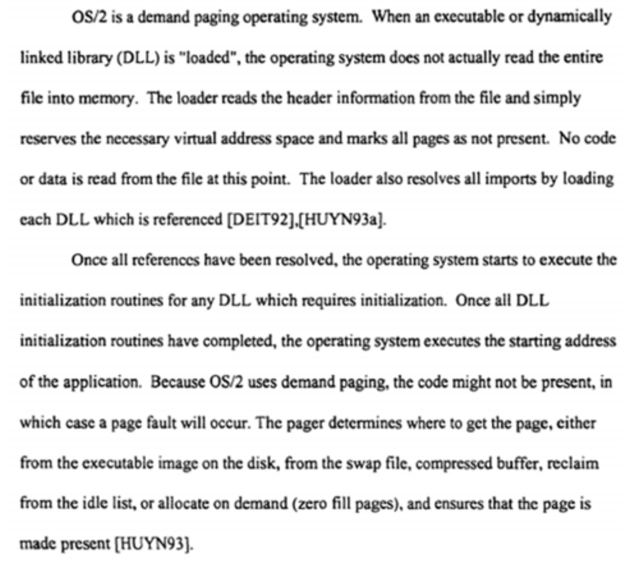
Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

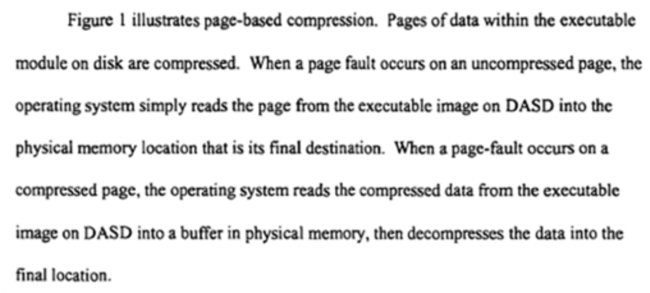
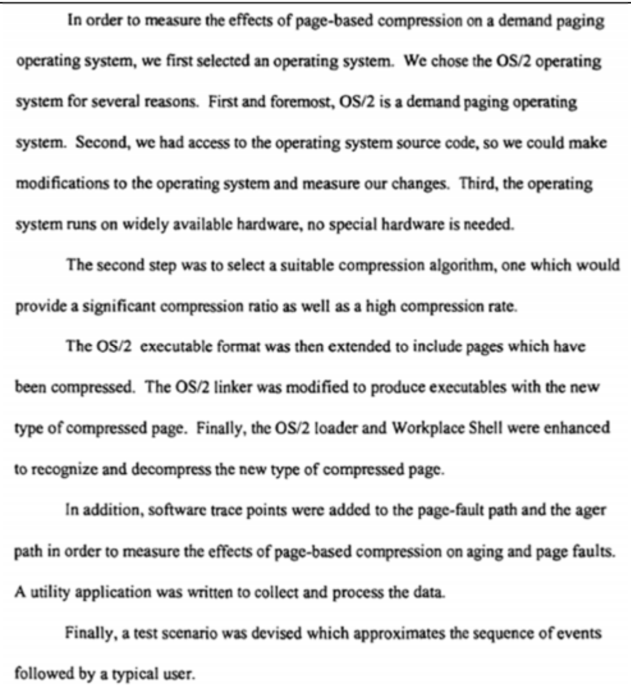


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

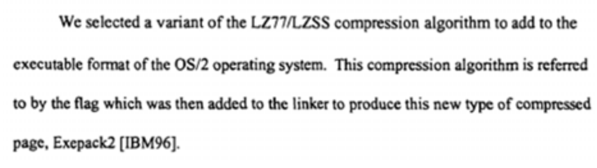
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

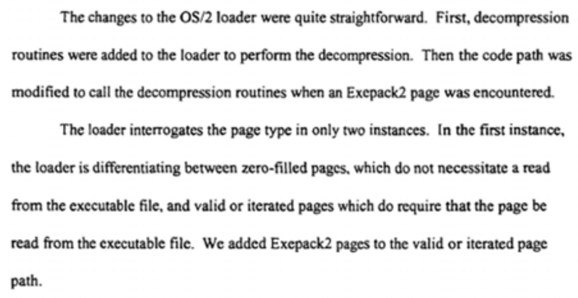
Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1



We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

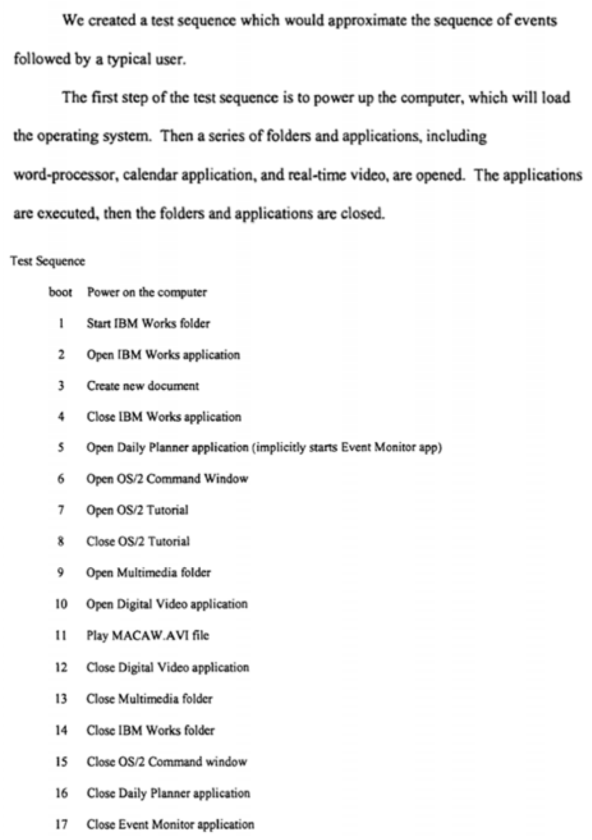
Wynn Thesis, 5.2



The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5



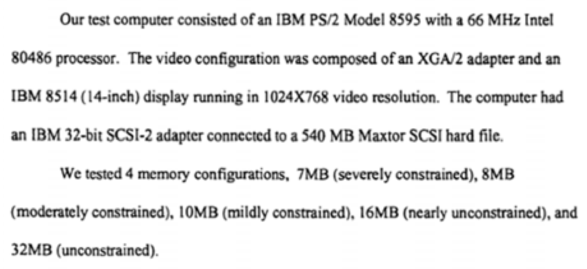
We created a test sequence which would approximate the sequence of events followed by a typical user.

The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

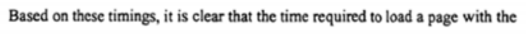
Wynn Thesis, 5.9



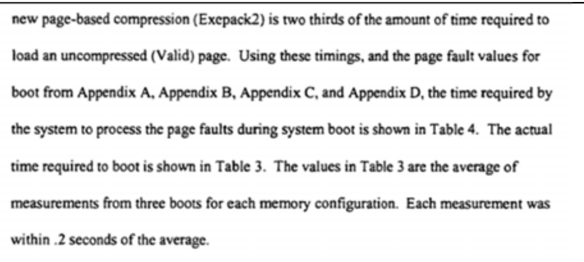
Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSI hard file.

We tested 4 memory configurations, 7MB (severely constrained), 8MB (moderately constrained), 10MB (mildly constrained), 16MB (nearly unconstrained), and 32MB (unconstrained).

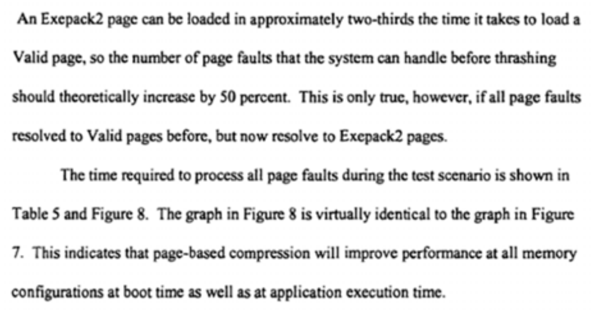
Wynn Thesis, 5.10



Based on these timings, it is clear that the time required to load a page with the

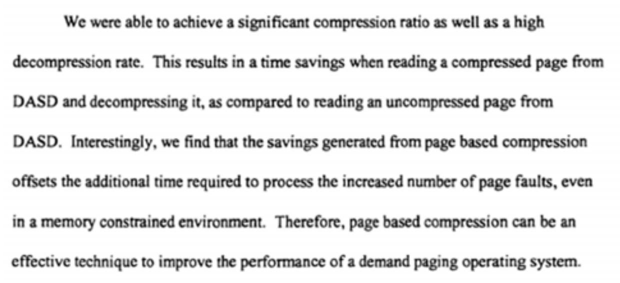

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

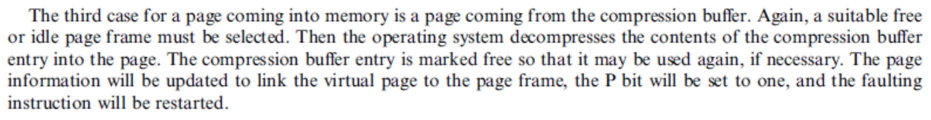
The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

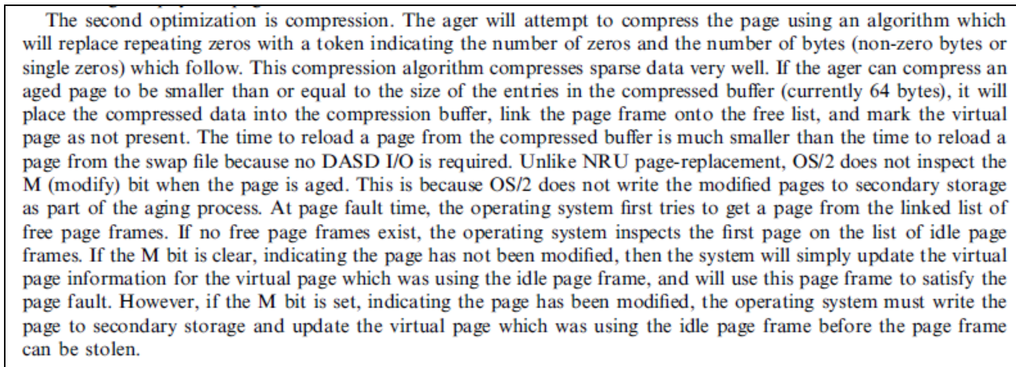

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7


The third case for a page coming into memory is a page coming from the compression buffer. Again, a suitable free or idle page frame must be selected. Then the operating system decompresses the contents of the compression buffer entry into the page. The compression buffer entry is marked free so that it may be used again, if necessary. The page information will be updated to link the virtual page to the page frame, the P bit will be set to one, and the faulting instruction will be restarted.

Wynn Article, 3.6



The second optimization is compression. The ager will attempt to compress the page using an algorithm which will replace repeating zeros with a token indicating the number of zeros and the number of bytes (non-zero bytes or single zeros) which follow. This compression algorithm compresses sparse data very well. If the ager can compress an aged page to be smaller than or equal to the size of the entries in the compressed buffer (currently 64 bytes), it will place the compressed data into the compression buffer, link the page frame onto the free list, and mark the virtual page as not present. The time to reload a page from the compressed buffer is much smaller than the time to reload a page from the swap file because no DASD I/O is required. Unlike NRU page-replacement, OS/2 does not inspect the M (modify) bit when the page is aged. This is because OS/2 does not write the modified pages to secondary storage as part of the aging process. At page fault time, the operating system first tries to get a page from the linked list of free page frames. If no free page frames exist, the operating system inspects the first page on the list of idle page frames. If the M bit is clear, indicating the page has not been modified, then the system will simply update the virtual page information for the virtual page which was using the idle page frame, and will use this page frame to satisfy the page fault. However, if the M bit is set, indicating the page has been modified, the operating system must write the page to secondary storage and update the virtual page which was using the idle page frame before the page frame can be stolen.

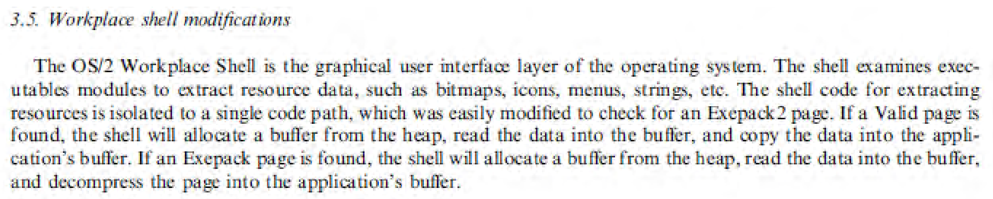
Wynn Article, 2.2

9.2 storing the additional portion of the operating system in the first memory, and	Wynn, as evidenced by the example citations below, discloses "storing the additional portion of the operating system in the first memory"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing the additional portion of the operating system in the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

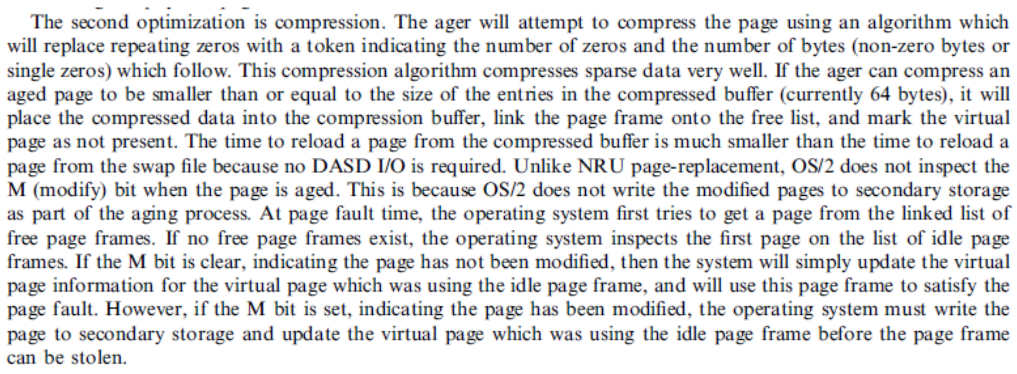
Wynn discloses this limitation:

See Claim 8.1 above.


3.5. Workplace shell modifications

The OS/2 Workplace Shell is the graphical user interface layer of the operating system. The shell examines executables modules to extract resource data, such as bitmaps, icons, menus, strings, etc. The shell code for extracting resources is isolated to a single code path, which was easily modified to check for an Exepack2 page. If a Valid page is found, the shell will allocate a buffer from the heap, read the data into the buffer, and copy the data into the application's buffer. If an Exepack page is found, the shell will allocate a buffer from the heap, read the data into the buffer, and decompress the page into the application's buffer.

Wynn Article 3.5


The second optimization is compression. The ager will attempt to compress the page using an algorithm which will replace repeating zeros with a token indicating the number of zeros and the number of bytes (non-zero bytes or single zeros) which follow. This compression algorithm compresses sparse data very well. If the ager can compress an aged page to be smaller than or equal to the size of the entries in the compressed buffer (currently 64 bytes), it will place the compressed data into the compression buffer, link the page frame onto the free list, and mark the virtual page as not present. The time to reload a page from the compressed buffer is much smaller than the time to reload a page from the swap file because no DASD I/O is required. Unlike NRU page-replacement, OS/2 does not inspect the M (modify) bit when the page is aged. This is because OS/2 does not write the modified pages to secondary storage as part of the aging process. At page fault time, the operating system first tries to get a page from the linked list of free page frames. If no free page frames exist, the operating system inspects the first page on the list of idle page frames. If the M bit is clear, indicating the page has not been modified, then the system will simply update the virtual page information for the virtual page which was using the idle page frame, and will use this page frame to satisfy the page fault. However, if the M bit is set, indicating the page has been modified, the operating system must write the page to secondary storage and update the virtual page which was using the idle page frame before the page frame can be stolen.

Wynn Article, 2.2

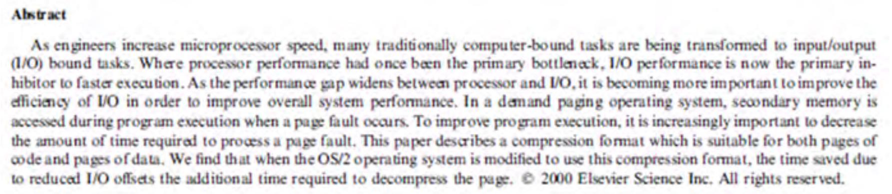
9.3 wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system.	Wynn, as evidenced by the example citations below, discloses "wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 8.5 above.	

10. The method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder.	Wynn, as evidenced by the example citations below, discloses "the method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

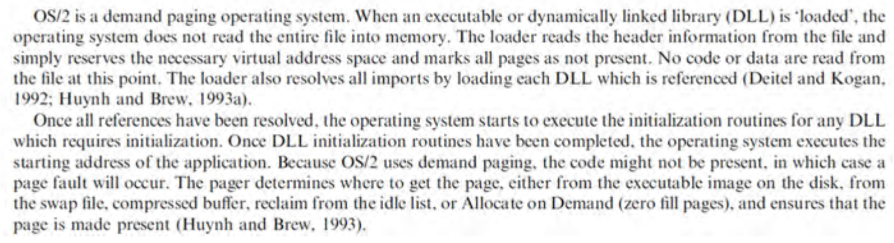
See Claim 9.1 above.



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

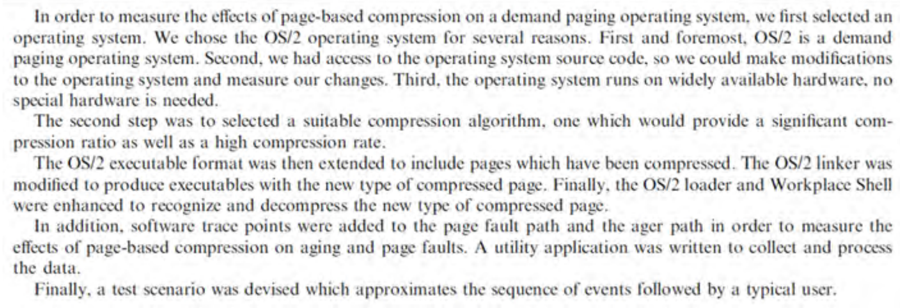
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

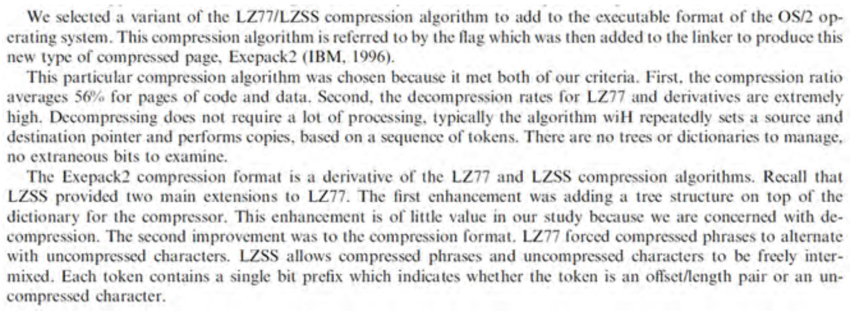
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

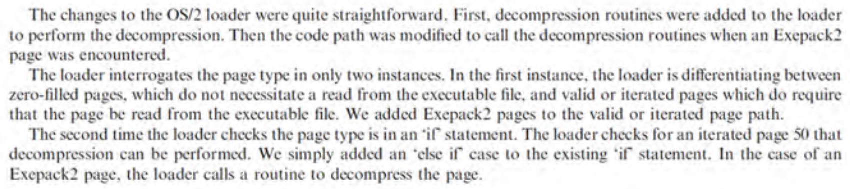


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

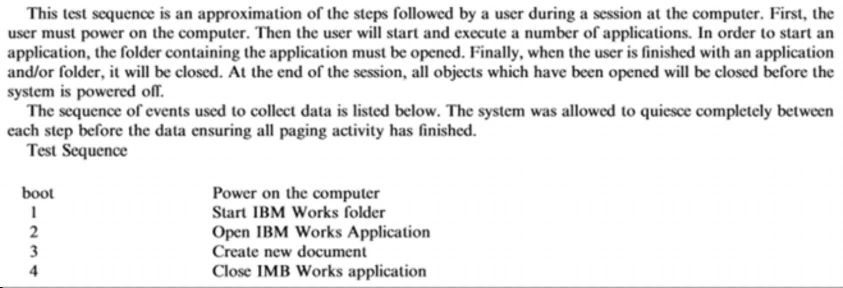


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

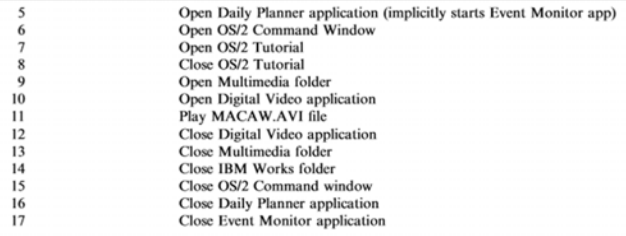


This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

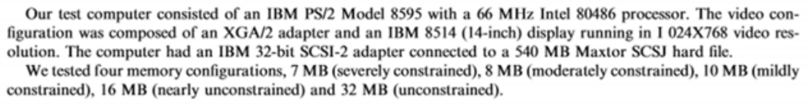
Test Sequence

boot	**Power on the computer**
1	**Start IBM Works folder**
2	**Open IBM Works Application**
3	**Create new document**
4	**Close IMB Works application**



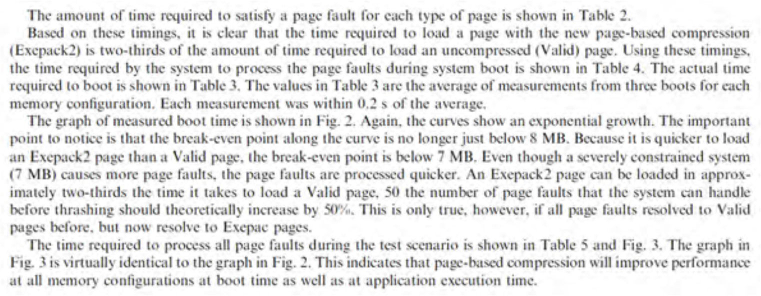
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Article, 3.8



Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in I 024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSJ hard file.
We tested four memory configurations, 7 MB (severely constrained), 8 MB (moderately constrained), 10 MB (mildly constrained), 16 MB (nearly unconstrained) and 32 MB (unconstrained).

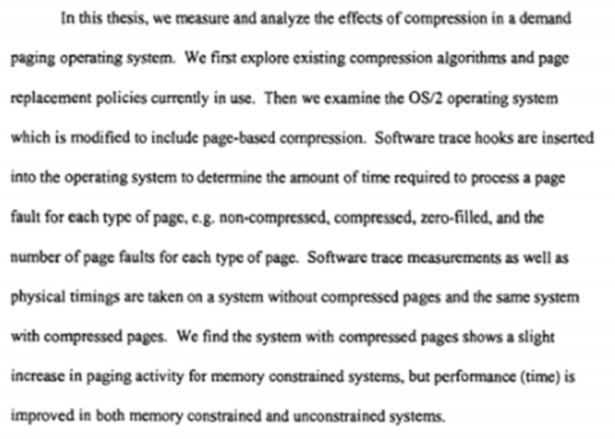
Wynn Article, 3.9



The amount of time required to satisfy a page fault for each type of page is shown in Table 2.
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Execpack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.
The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.
The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

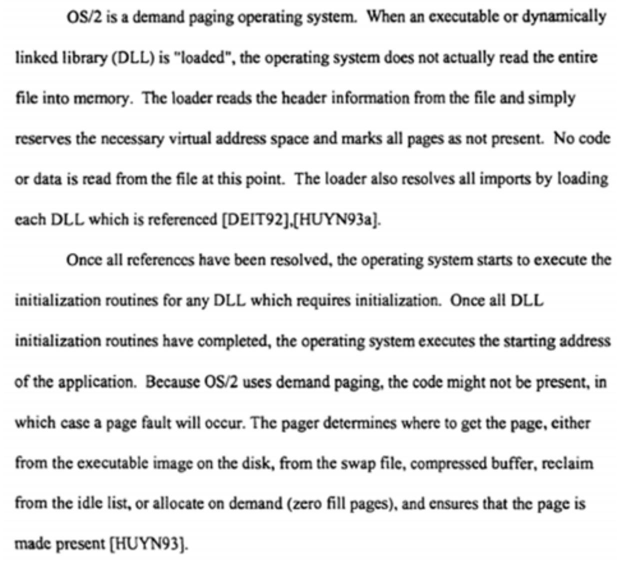
Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

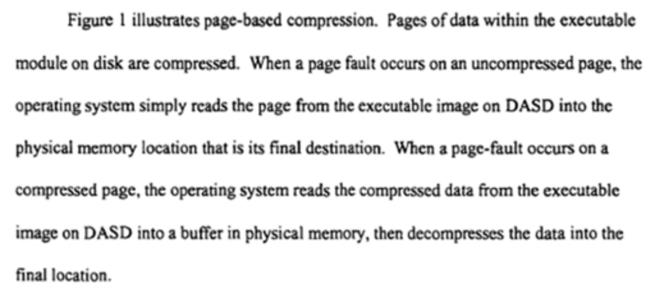
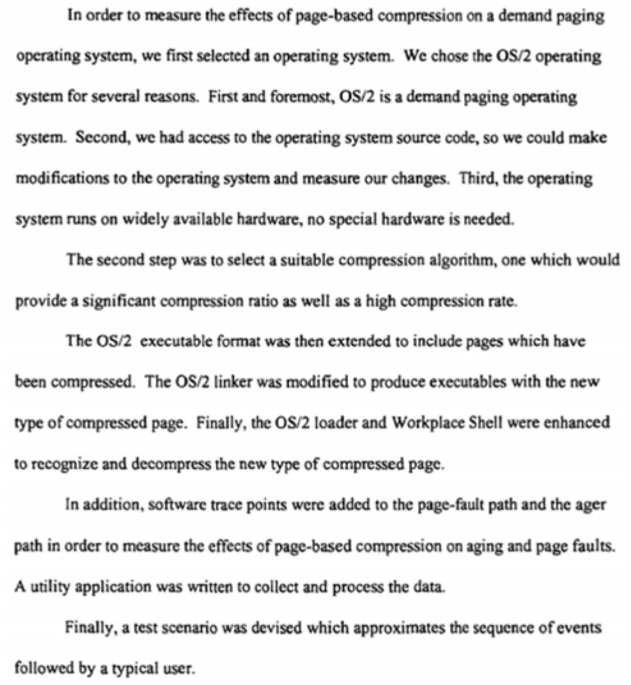


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

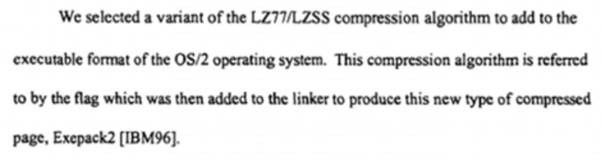
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

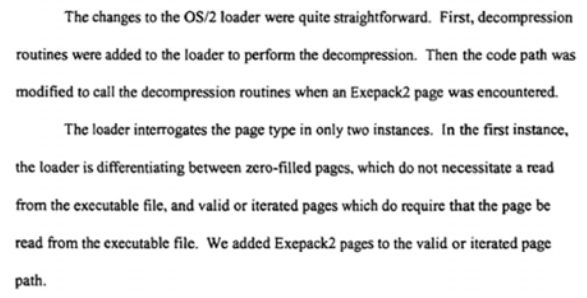
Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1



We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

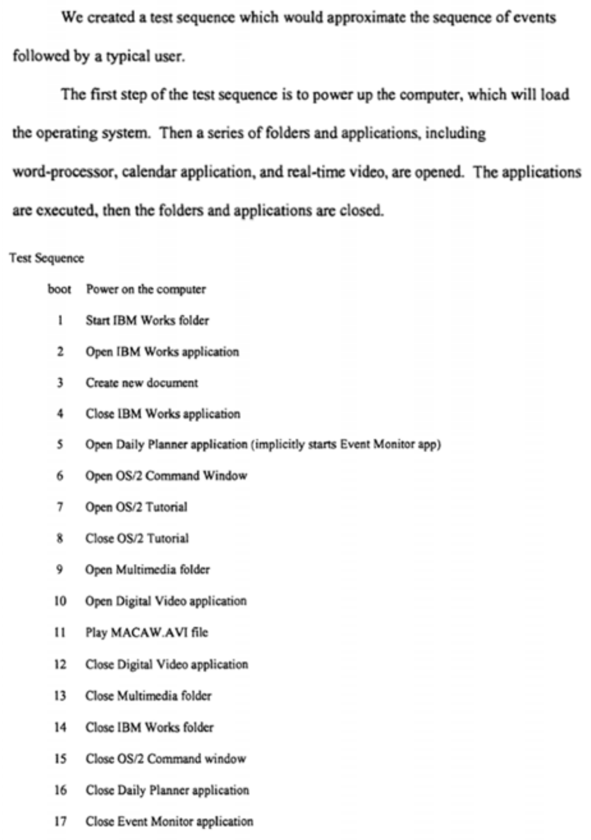
Wynn Thesis, 5.2



The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5

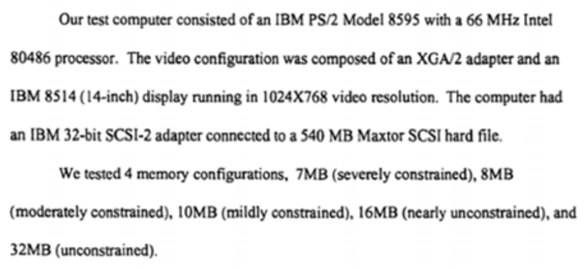

We created a test sequence which would approximate the sequence of events followed by a typical user.

The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.

Test Sequence

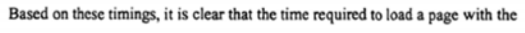
boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

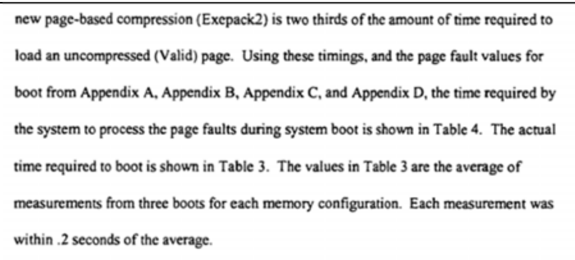
Wynn Thesis, 5.9


Our test computer consisted of an IBM PS/2 Model 8595 with a 66 MHz Intel 80486 processor. The video configuration was composed of an XGA/2 adapter and an IBM 8514 (14-inch) display running in 1024X768 video resolution. The computer had an IBM 32-bit SCSI-2 adapter connected to a 540 MB Maxtor SCSI hard file.

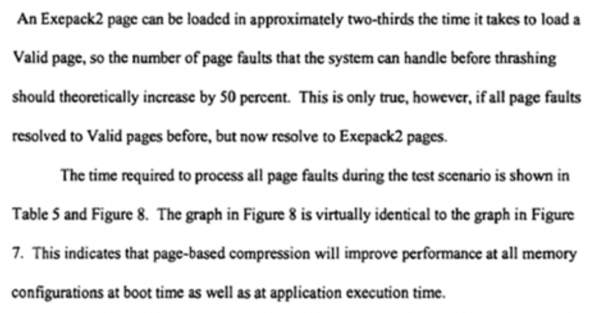
We tested 4 memory configurations, 7MB (severely constrained), 8MB (moderately constrained), 10MB (mildly constrained), 16MB (nearly unconstrained), and 32MB (unconstrained).

Wynn Thesis, 5.10


Based on these timings, it is clear that the time required to load a page with the

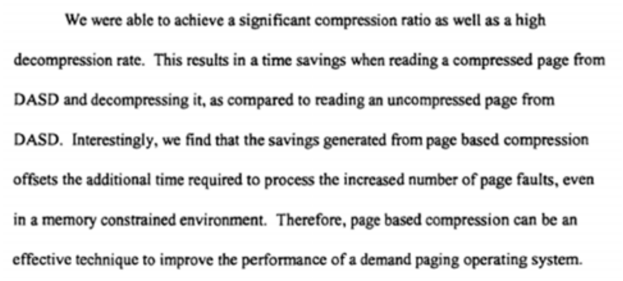

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

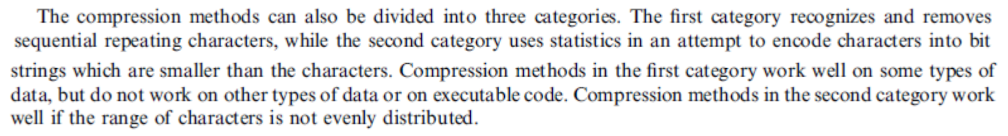
The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3

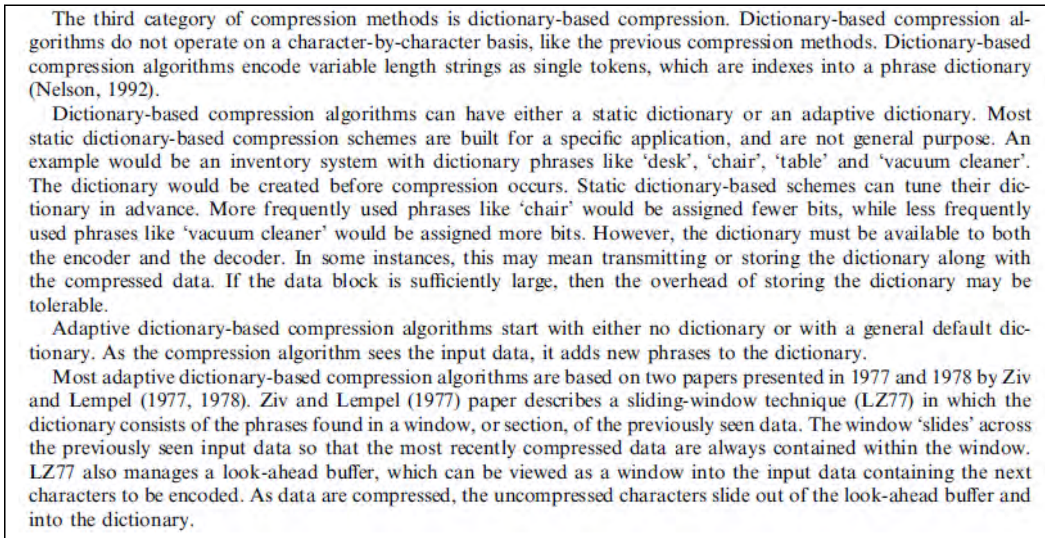

We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7


The compression methods can also be divided into three categories. The first category recognizes and removes sequential repeating characters, while the second category uses statistics in an attempt to encode characters into bit strings which are smaller than the characters. Compression methods in the first category work well on some types of data, but do not work on other types of data or on executable code. Compression methods in the second category work well if the range of characters is not evenly distributed.

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn



The third category of compression methods is dictionary-based compression. Dictionary-based compression algorithms do not operate on a character-by-character basis, like the previous compression methods. Dictionary-based compression algorithms encode variable length strings as single tokens, which are indexes into a phrase dictionary (Nelson, 1992).

Dictionary-based compression algorithms can have either a static dictionary or an adaptive dictionary. Most static dictionary-based compression schemes are built for a specific application, and are not general purpose. An example would be an inventory system with dictionary phrases like 'desk', 'chair', 'table' and 'vacuum cleaner'. The dictionary would be created before compression occurs. Static dictionary-based schemes can tune their dictionary in advance. More frequently used phrases like 'chair' would be assigned fewer bits, while less frequently used phrases like 'vacuum cleaner' would be assigned more bits. However, the dictionary must be available to both the encoder and the decoder. In some instances, this may mean transmitting or storing the dictionary along with the compressed data. If the data block is sufficiently large, then the overhead of storing the dictionary may be tolerable.

Adaptive dictionary-based compression algorithms start with either no dictionary or with a general default dictionary. As the compression algorithm sees the input data, it adds new phrases to the dictionary.

Most adaptive dictionary-based compression algorithms are based on two papers presented in 1977 and 1978 by Ziv and Lempel (1977, 1978). Ziv and Lempel (1977) paper describes a sliding-window technique (LZ77) in which the dictionary consists of the phrases found in a window, or section, of the previously seen data. The window 'slides' across the previously seen input data so that the most recently compressed data are always contained within the window. LZ77 also manages a look-ahead buffer, which can be viewed as a window into the input data containing the next characters to be encoded. As data are compressed, the uncompressed characters slide out of the look-ahead buffer and into the dictionary.

Wynn Article, 2.1

11 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Wynn, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1 (Preamble) above.	

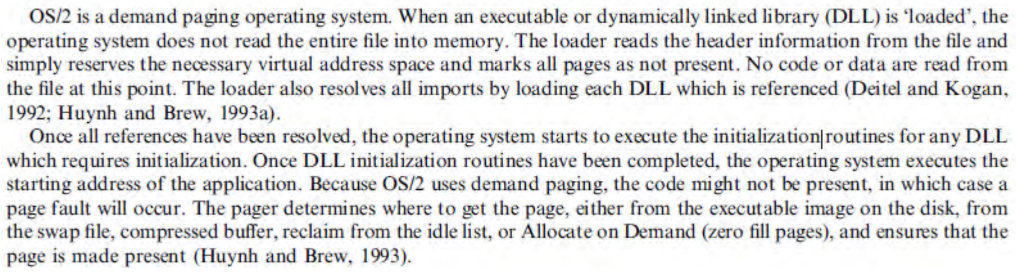
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

11.1 loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system;	Wynn, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

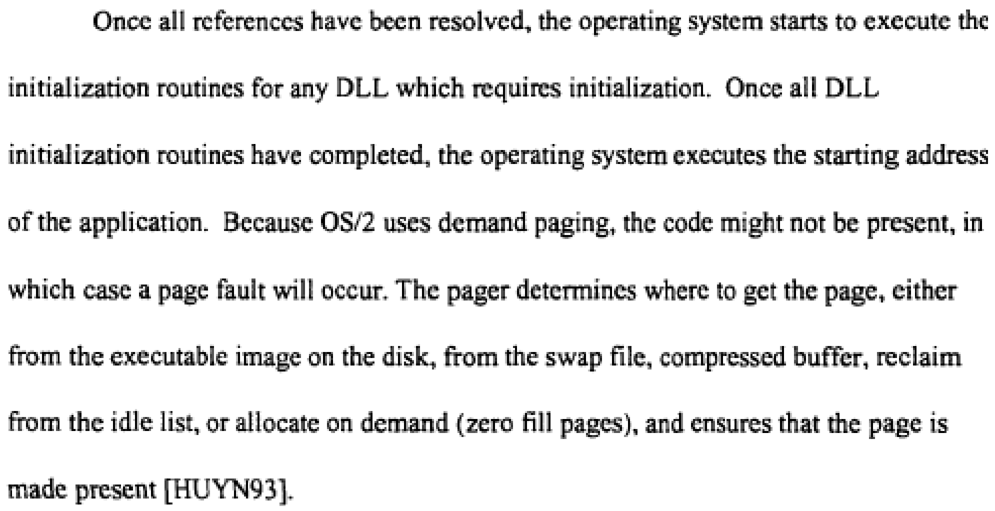
Wynn discloses this limitation:

See Claim 1.1 and 6(Preamble) above.


OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization|routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2


Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Article, 4.2

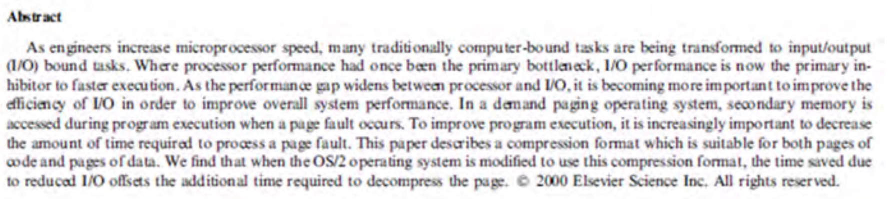
11.2 accessing the loaded boot data in compressed form from the memory;	Wynn, as evidenced by the example citations below, discloses "accessing the loaded boot data in compressed form from the memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in compressed form from the memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.2 above.	

11.3 decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Wynn, as evidenced by the example citations below, discloses "decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.3 above.	

11.4 utilizing the decompressed boot data to load at least a portion of the operating system for the computer system; and	Wynn, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to load at least a portion of the operating system for the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to load at least a portion of the operating system for the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

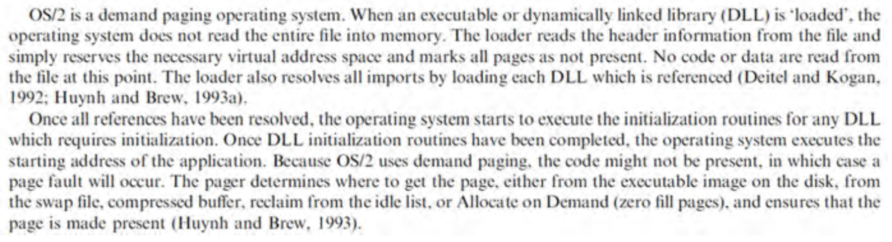
Wynn discloses this limitation:



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

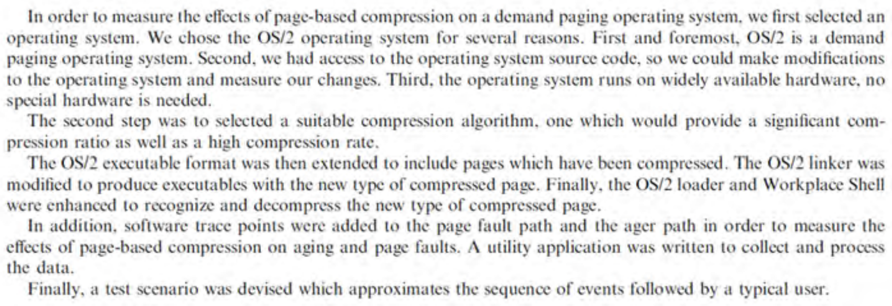
Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

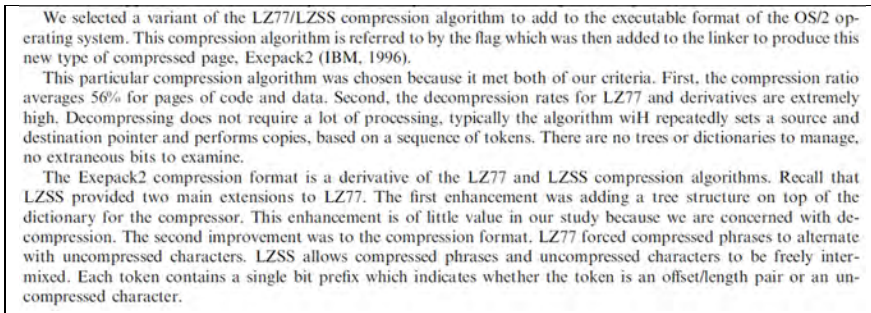
The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

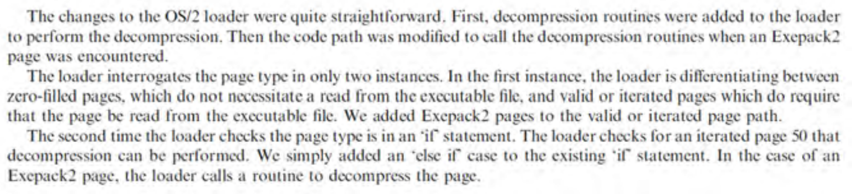
Wynn Article, 3.1


We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

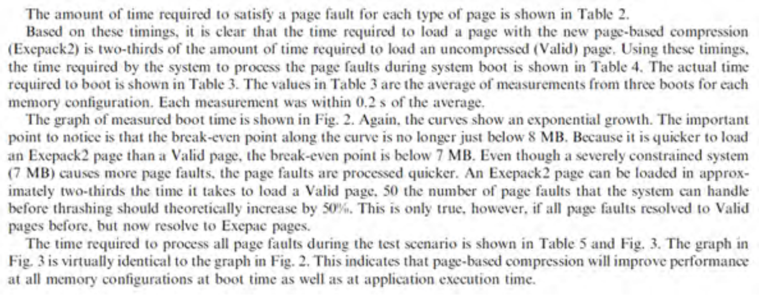
Wynn Article, 3.2


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4


The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

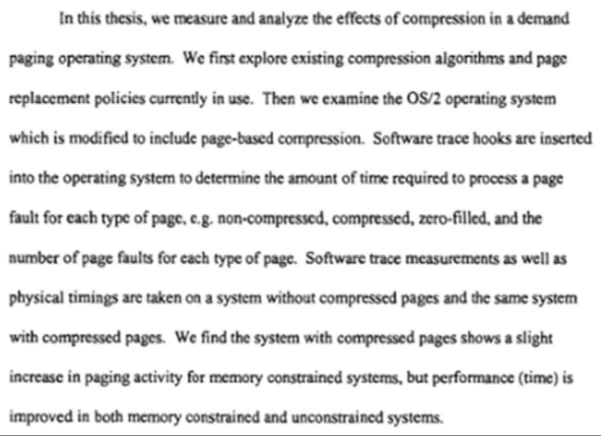
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

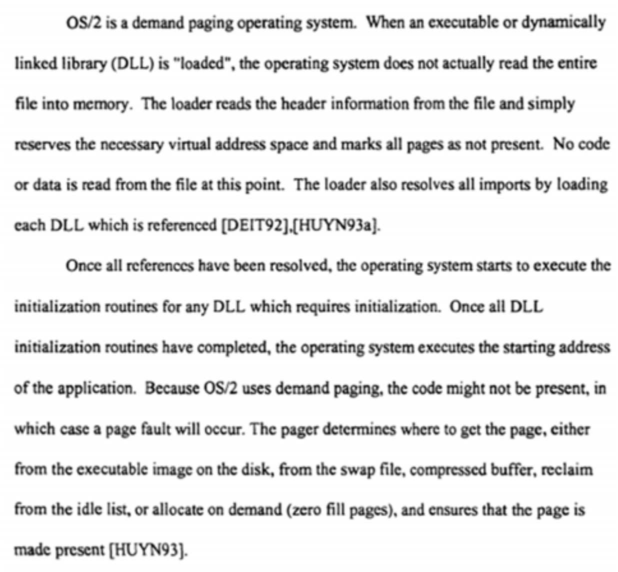
The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2


In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

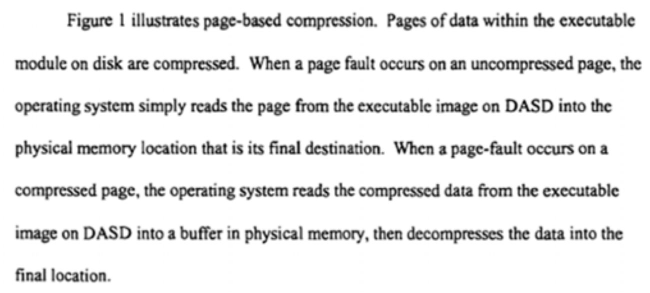
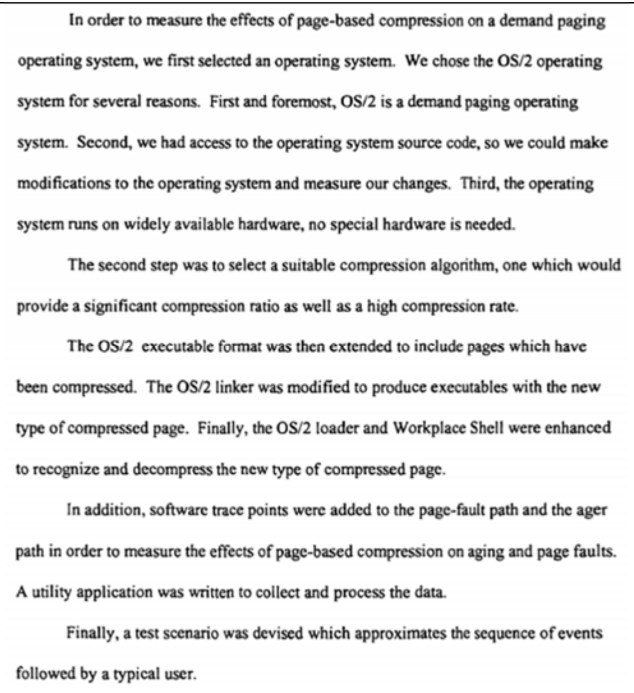


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

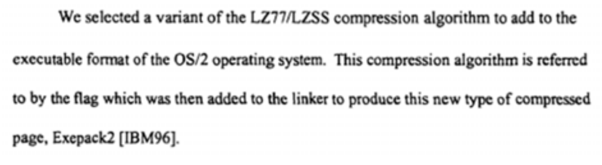
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

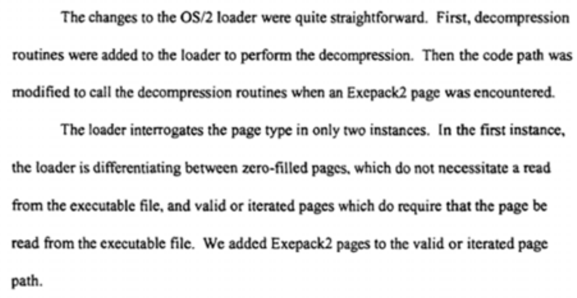
Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1



We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

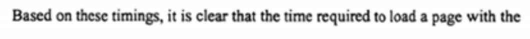
Wynn Thesis, 5.2



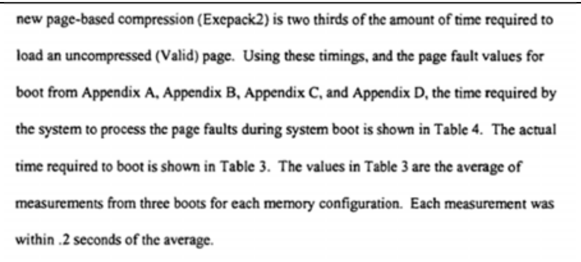
The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

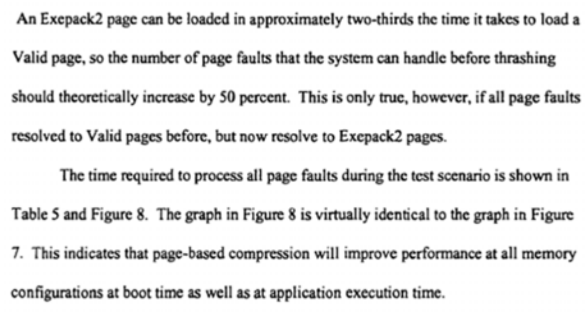
Wynn Thesis, 5.5



Based on these timings, it is clear that the time required to load a page with the

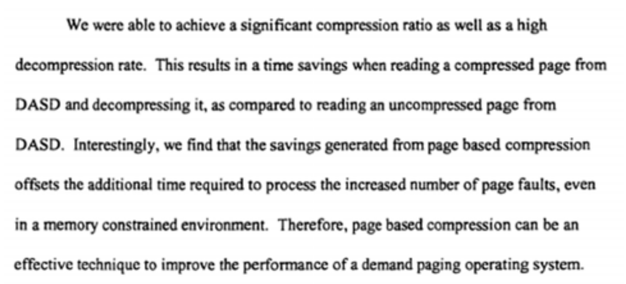

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3


We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

11.5 updating the boot data list.	Wynn, as evidenced by the example citations below, discloses “updating the boot data list”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4.1 above.	

13 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Wynn, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1 (Preamble) above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

13.1 loading boot data in a compressed form that is associated with a boot data list from a boot device;	Wynn, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 above.	

13.2 accessing the loaded boot data in the compressed form;	Wynn, as evidenced by the example citations below, discloses "accessing the loaded boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.2 above.	

13.3 decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Wynn, as evidenced by the example citations below, discloses "decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.3 above.	

13.4 updating the boot data list.	Wynn, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

14 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Wynn, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1 (Preamble) above.	

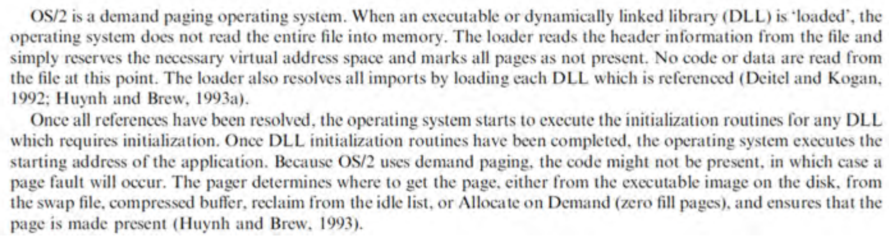
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

14.1 accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list;	Wynn, as evidenced by the example citations below, discloses "accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

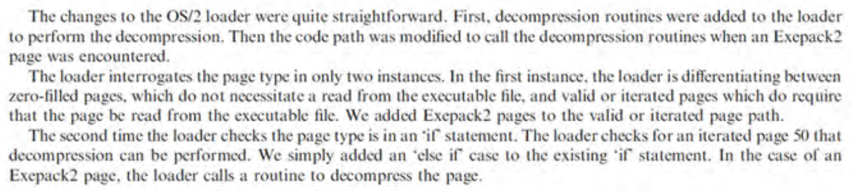
See Claims 1.1 and 1.2 above.



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

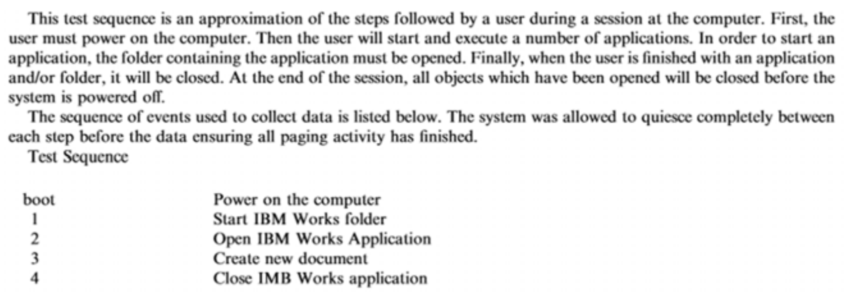


The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

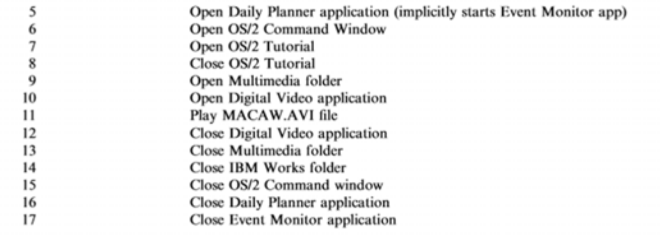


This test sequence is an approximation of the steps followed by a user during a session at the computer. First, the user must power on the computer. Then the user will start and execute a number of applications. In order to start an application, the folder containing the application must be opened. Finally, when the user is finished with an application and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed before the system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely between each step before the data ensuring all paging activity has finished.

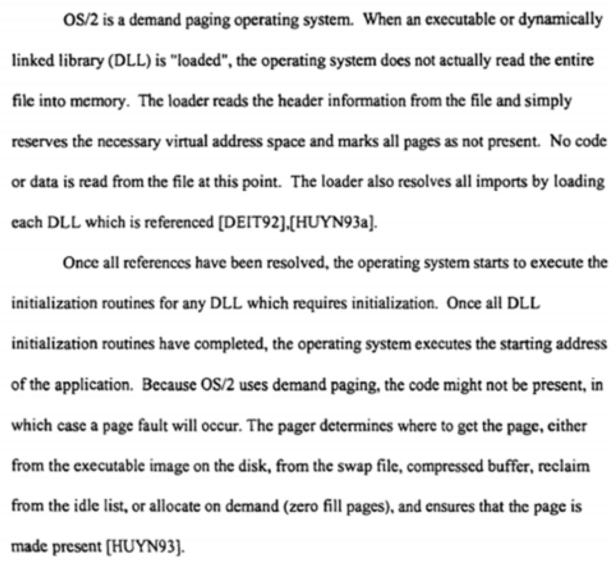
Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application



5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

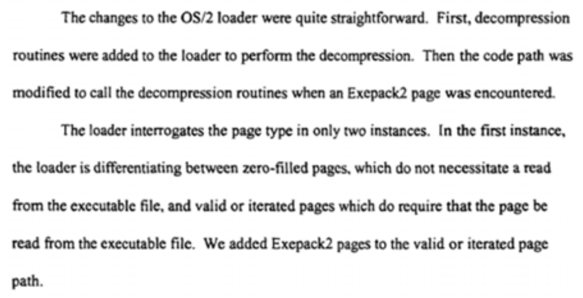
Wynn Article, 3.8



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

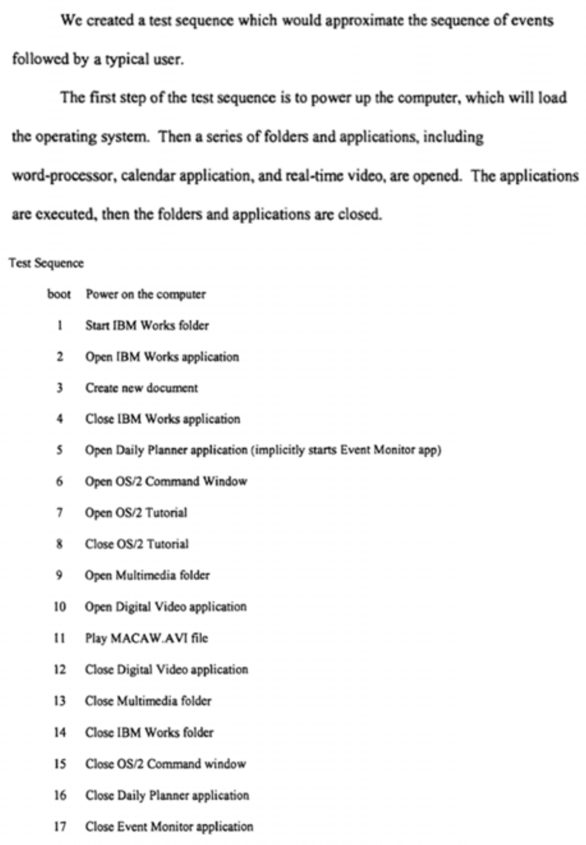
Wynn Thesis, 4.2



The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

Wynn Thesis, 5.5



We created a test sequence which would approximate the sequence of events followed by a typical user.

The first step of the test sequence is to power up the computer, which will load the operating system. Then a series of folders and applications, including word-processor, calendar application, and real-time video, are opened. The applications are executed, then the folders and applications are closed.

Test Sequence

boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works application
3	Create new document
4	Close IBM Works application
5	Open Daily Planner application (implicitly starts Event Monitor app)
6	Open OS/2 Command Window
7	Open OS/2 Tutorial
8	Close OS/2 Tutorial
9	Open Multimedia folder
10	Open Digital Video application
11	Play MACAW.AVI file
12	Close Digital Video application
13	Close Multimedia folder
14	Close IBM Works folder
15	Close OS/2 Command window
16	Close Daily Planner application
17	Close Event Monitor application

Wynn Thesis, 5.9

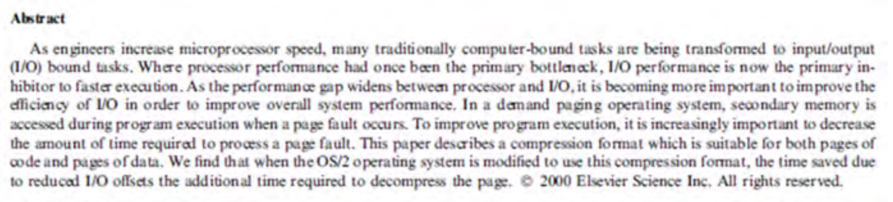
14.2 loading the boot data into a memory; and	Wynn, as evidenced by the example citations below, discloses "loading the boot data into a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the boot data into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.1 above.	

14.3 servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form; and	Wynn, as evidenced by the example citations below, discloses "servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

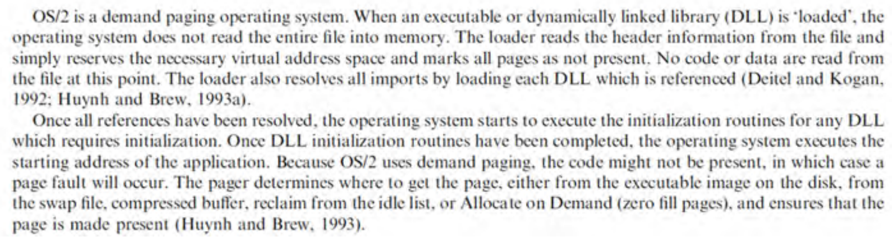
See Claims 1.2 and 1.3 above.



Abstract

As engineers increase microprocessor speed, many traditionally computer-bound tasks are being transformed to input/output (I/O) bound tasks. Where processor performance had once been the primary bottleneck, I/O performance is now the primary inhibitor to faster execution. As the performance gap widens between processor and I/O, it is becoming more important to improve the efficiency of I/O in order to improve overall system performance. In a demand paging operating system, secondary memory is accessed during program execution when a page fault occurs. To improve program execution, it is increasingly important to decrease the amount of time required to process a page fault. This paper describes a compression format which is suitable for both pages of code and pages of data. We find that when the OS/2 operating system is modified to use this compression format, the time saved due to reduced I/O offsets the additional time required to decompress the page. © 2000 Elsevier Science Inc. All rights reserved.

Wynn Article, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once DLL initialization routines have been completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or Allocate on Demand (zero fill pages), and ensures that the page is made present (Huynh and Brew, 1993).

Wynn Article, 2.2

In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

The second step was to selected a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Article, 3.1

We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 (IBM, 1996).

This particular compression algorithm was chosen because it met both of our criteria. First, the compression ratio averages 56% for pages of code and data. Second, the decompression rates for LZ77 and derivatives are extremely high. Decompressing does not require a lot of processing, typically the algorithm wiH repeatedly sets a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

The Exepack2 compression format is a derivative of the LZ77 and LZSS compression algorithms. Recall that LZSS provided two main extensions to LZ77. The first enhancement was adding a tree structure on top of the dictionary for the compressor. This enhancement is of little value in our study because we are concerned with decompression. The second improvement was to the compression format. LZ77 forced compressed phrases to alternate with uncompressed characters. LZSS allows compressed phrases and uncompressed characters to be freely intermixed. Each token contains a single bit prefix which indicates whether the token is an offset/length pair or an uncompressed character.

Wynn Article, 3.2

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article, 3.4

The amount of time required to satisfy a page fault for each type of page is shown in Table 2.

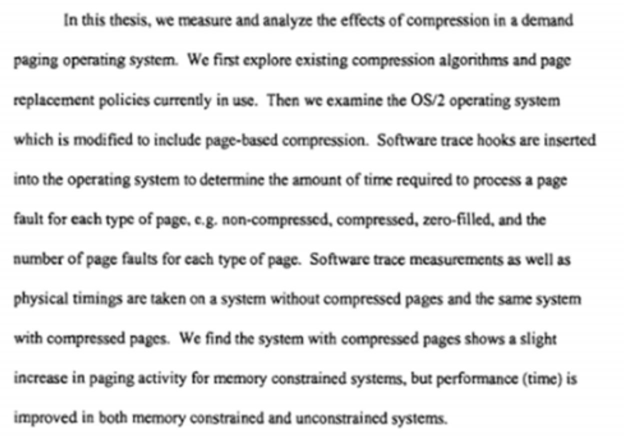
Based on these timings, it is clear that the time required to load a page with the new page-based compression (Exepack2) is two-thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within 0.2 s of the average.

The graph of measured boot time is shown in Fig. 2. Again, the curves show an exponential growth. The important point to notice is that the break-even point along the curve is no longer just below 8 MB. Because it is quicker to load an Exepack2 page than a Valid page, the break-even point is below 7 MB. Even though a severely constrained system (7 MB) causes more page faults, the page faults are processed quicker. An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, 50 the number of page faults that the system can handle before thrashing should theoretically increase by 50%. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepac pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Fig. 3. The graph in Fig. 3 is virtually identical to the graph in Fig. 2. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

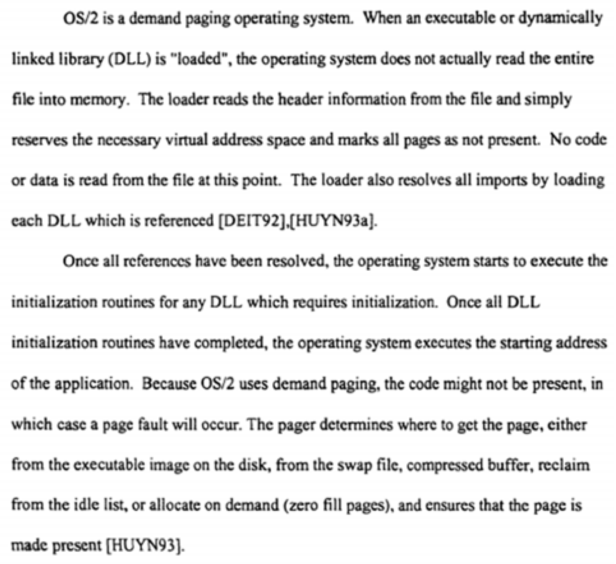
Wynn Article, 4.3

See also, Wynn Article, Table 3 and Fig. 2



In this thesis, we measure and analyze the effects of compression in a demand paging operating system. We first explore existing compression algorithms and page replacement policies currently in use. Then we examine the OS/2 operating system which is modified to include page-based compression. Software trace hooks are inserted into the operating system to determine the amount of time required to process a page fault for each type of page, e.g. non-compressed, compressed, zero-filled, and the number of page faults for each type of page. Software trace measurements as well as physical timings are taken on a system without compressed pages and the same system with compressed pages. We find the system with compressed pages shows a slight increase in paging activity for memory constrained systems, but performance (time) is improved in both memory constrained and unconstrained systems.

Wynn Thesis, Abstract



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is "loaded", the operating system does not actually read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data is read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced [DEIT92],[HUYN93a].

Once all references have been resolved, the operating system starts to execute the initialization routines for any DLL which requires initialization. Once all DLL initialization routines have completed, the operating system executes the starting address of the application. Because OS/2 uses demand paging, the code might not be present, in which case a page fault will occur. The pager determines where to get the page, either from the executable image on the disk, from the swap file, compressed buffer, reclaim from the idle list, or allocate on demand (zero fill pages), and ensures that the page is made present [HUYN93].

Wynn Thesis, 4.2

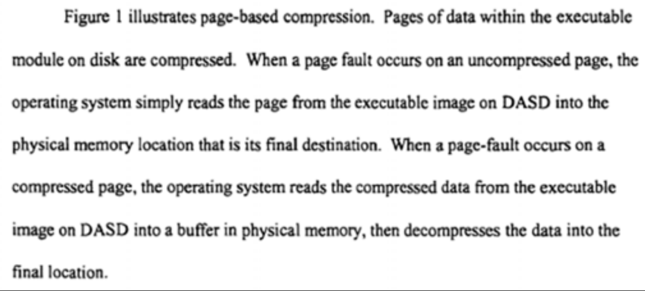
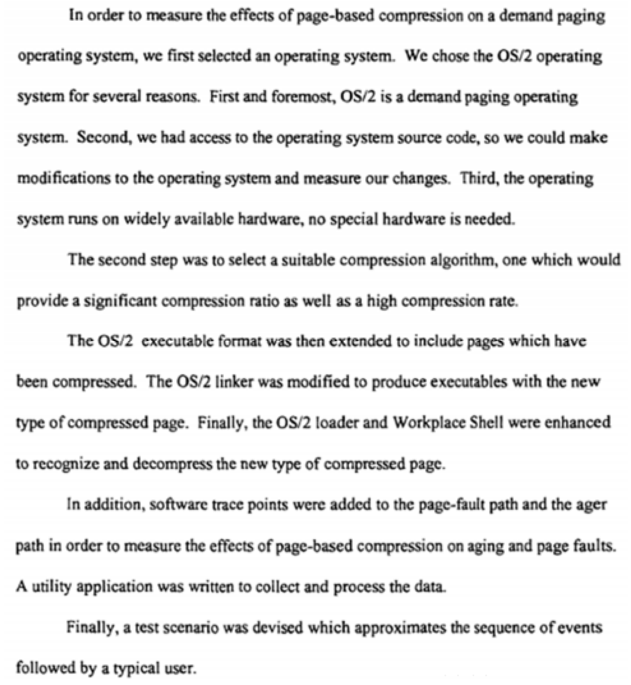


Figure 1 illustrates page-based compression. Pages of data within the executable module on disk are compressed. When a page fault occurs on an uncompressed page, the operating system simply reads the page from the executable image on DASD into the physical memory location that is its final destination. When a page-fault occurs on a compressed page, the operating system reads the compressed data from the executable image on DASD into a buffer in physical memory, then decompresses the data into the final location.



In order to measure the effects of page-based compression on a demand paging operating system, we first selected an operating system. We chose the OS/2 operating system for several reasons. First and foremost, OS/2 is a demand paging operating system. Second, we had access to the operating system source code, so we could make modifications to the operating system and measure our changes. Third, the operating system runs on widely available hardware, no special hardware is needed.

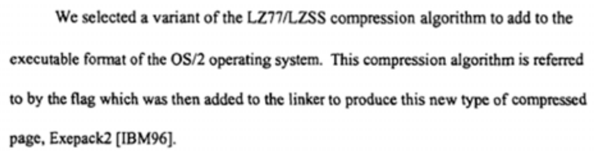
The second step was to select a suitable compression algorithm, one which would provide a significant compression ratio as well as a high compression rate.

The OS/2 executable format was then extended to include pages which have been compressed. The OS/2 linker was modified to produce executables with the new type of compressed page. Finally, the OS/2 loader and Workplace Shell were enhanced to recognize and decompress the new type of compressed page.

In addition, software trace points were added to the page-fault path and the ager path in order to measure the effects of page-based compression on aging and page faults. A utility application was written to collect and process the data.

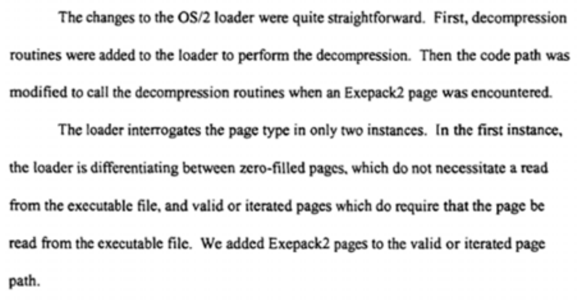
Finally, a test scenario was devised which approximates the sequence of events followed by a typical user.

Wynn Thesis, 5.1



We selected a variant of the LZ77/LZSS compression algorithm to add to the executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

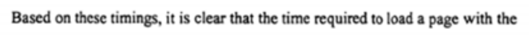
Wynn Thesis, 5.2



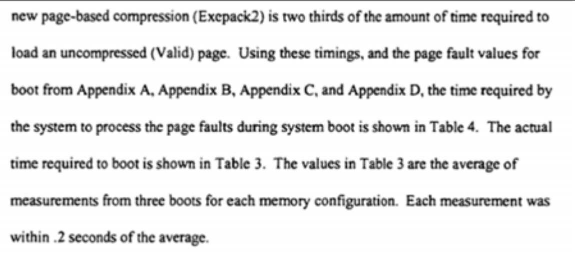
The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

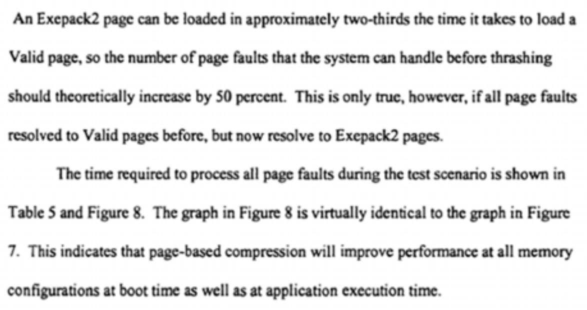
Wynn Thesis, 5.5



Based on these timings, it is clear that the time required to load a page with the

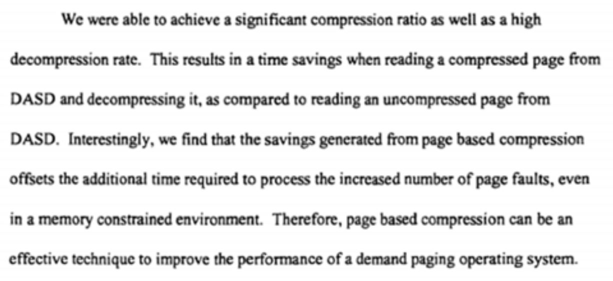

new page-based compression (Exepack2) is two thirds of the amount of time required to load an uncompressed (Valid) page. Using these timings, and the page fault values for boot from Appendix A, Appendix B, Appendix C, and Appendix D, the time required by the system to process the page faults during system boot is shown in Table 4. The actual time required to boot is shown in Table 3. The values in Table 3 are the average of measurements from three boots for each memory configuration. Each measurement was within .2 seconds of the average.

Wynn Thesis, 6.3


An Exepack2 page can be loaded in approximately two-thirds the time it takes to load a Valid page, so the number of page faults that the system can handle before thrashing should theoretically increase by 50 percent. This is only true, however, if all page faults resolved to Valid pages before, but now resolve to Exepack2 pages.

The time required to process all page faults during the test scenario is shown in Table 5 and Figure 8. The graph in Figure 8 is virtually identical to the graph in Figure 7. This indicates that page-based compression will improve performance at all memory configurations at boot time as well as at application execution time.

Wynn Thesis, 6.3


We were able to achieve a significant compression ratio as well as a high decompression rate. This results in a time savings when reading a compressed page from DASD and decompressing it, as compared to reading an uncompressed page from DASD. Interestingly, we find that the savings generated from page based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page based compression can be an effective technique to improve the performance of a demand paging operating system.

Wynn Thesis, 7.2

See also Wynn Thesis, Table 2-4, Fig. 7

14.4 updating the boot data list.	Wynn, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 1.4.1 above.	

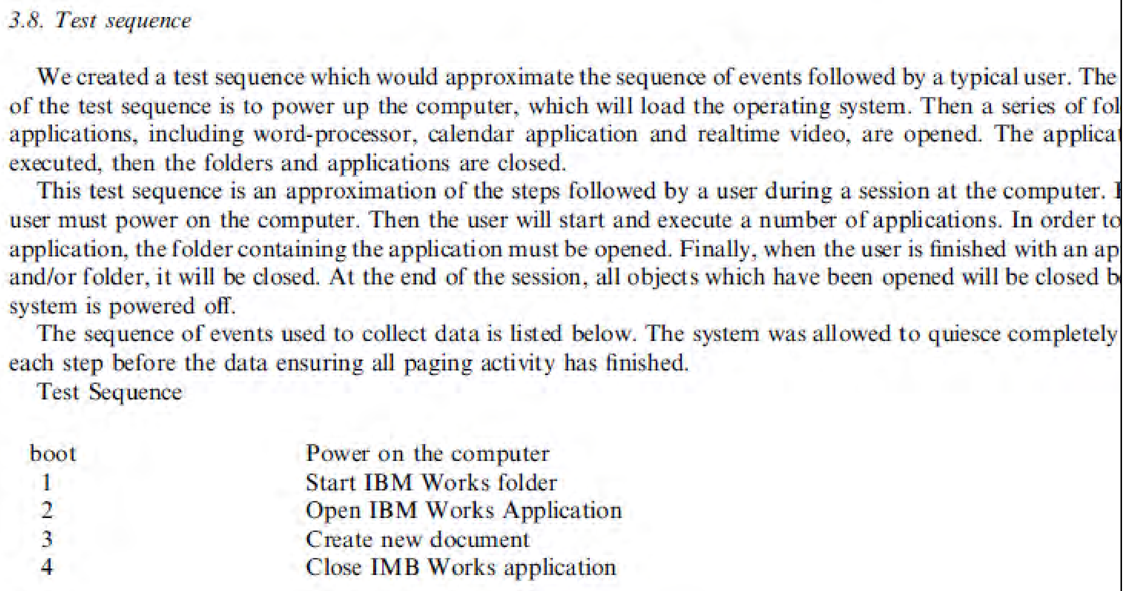
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

15. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system.	Wynn, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1 (Preamble) and 1.1 above.



3.8. Test sequence

We created a test sequence which would approximate the sequence of events followed by a typical user. The of the test sequence is to power up the computer, which will load the operating system. Then a series of fol applications, including word-processor, calendar application and realtime video, are opened. The applicat executed, then the folders and applications are closed.

This test sequence is an approximation of the steps followed by a user during a session at the computer. I user must power on the computer. Then the user will start and execute a number of applications. In order to application, the folder containing the application must be opened. Finally, when the user is finished with an ap and/or folder, it will be closed. At the end of the session, all objects which have been opened will be closed b system is powered off.

The sequence of events used to collect data is listed below. The system was allowed to quiesce completely each step before the data ensuring all paging activity has finished.

Test Sequence

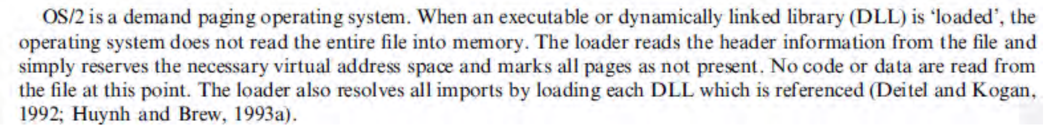
boot	Power on the computer
1	Start IBM Works folder
2	Open IBM Works Application
3	Create new document
4	Close IMB Works application

Wynn, 3.8

16. The method of claim 14, wherein the operating system comprises: a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 14, wherein the operating system comprises: a plurality of files."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:



OS/2 is a demand paging operating system. When an executable or dynamically linked library (DLL) is 'loaded', the operating system does not read the entire file into memory. The loader reads the header information from the file and simply reserves the necessary virtual address space and marks all pages as not present. No code or data are read from the file at this point. The loader also resolves all imports by loading each DLL which is referenced (Deitel and Kogan, 1992; Huynh and Brew, 1993a).

Wynn Article, 2.2

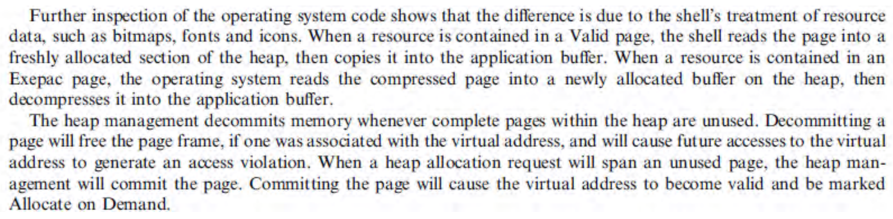
17. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.	Wynn, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 15 above.	

19.1 The method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:	Wynn, as evidenced by the example citations below, discloses "the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.1 and 1.2 above



Further inspection of the operating system code shows that the difference is due to the shell's treatment of resource data, such as bitmaps, fonts and icons. When a resource is contained in a Valid page, the shell reads the page into a freshly allocated section of the heap, then copies it into the application buffer. When a resource is contained in an Exepac page, the operating system reads the compressed page into a newly allocated buffer on the heap, then decompresses it into the application buffer.

The heap management decommits memory whenever complete pages within the heap are unused. Decommitting a page will free the page frame, if one was associated with the virtual address, and will cause future accesses to the virtual address to generate an access violation. When a heap allocation request will span an unused page, the heap management will commit the page. Committing the page will cause the virtual address to become valid and be marked Allocate on Demand.

Wynn Article, 4.2

19.2 associating the accessed boot data that is not associated with the boot data list to the boot data list.	Wynn, as evidenced by the example citations below, discloses "associating the accessed boot data that is not associated with the boot data list to the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, associating the accessed boot data that is not associated with the boot data list to the boot data list.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 2 above.	

23. The method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 16 above.	

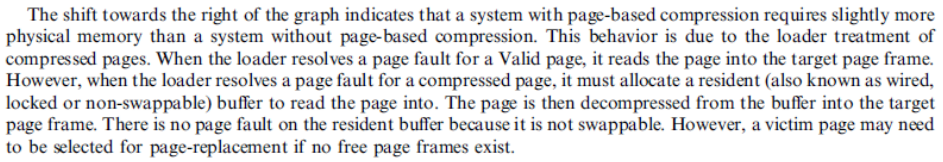
24. The method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 15 above.	

27. The method of claim 1, wherein the memory comprises: a physical memory.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein the memory comprises: a physical memory."

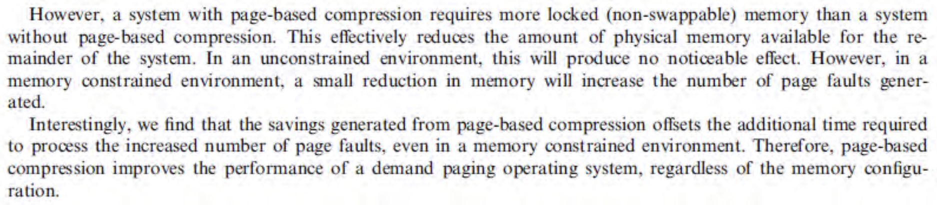
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.1 and 1.2 above


The shift towards the right of the graph indicates that a system with page-based compression requires slightly more physical memory than a system without page-based compression. This behavior is due to the loader treatment of compressed pages. When the loader resolves a page fault for a Valid page, it reads the page into the target page frame. However, when the loader resolves a page fault for a compressed page, it must allocate a resident (also known as wired, locked or non-swappable) buffer to read the page into. The page is then decompressed from the buffer into the target page frame. There is no page fault on the resident buffer because it is not swappable. However, a victim page may need to be selected for page-replacement if no free page frames exist.

Wynn Article, 4.2


However, a system with page-based compression requires more locked (non-swappable) memory than a system without page-based compression. This effectively reduces the amount of physical memory available for the remainder of the system. In an unconstrained environment, this will produce no noticeable effect. However, in a memory constrained environment, a small reduction in memory will increase the number of page faults generated.

Interestingly, we find that the savings generated from page-based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page-based compression improves the performance of a demand paging operating system, regardless of the memory configuration.

Wynn Article, 5.2

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

28. The method of claim 1, wherein the operating system comprises: a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 16 above.	

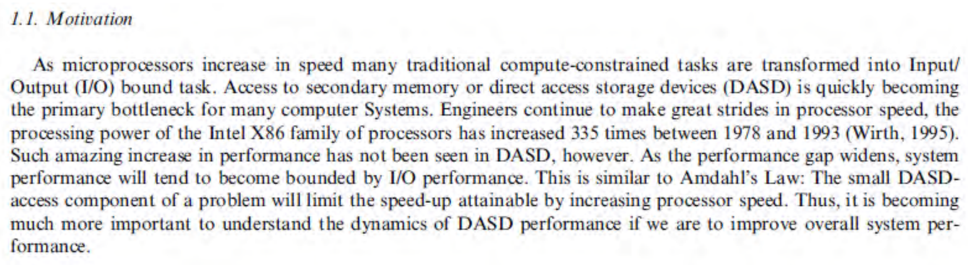
29. The method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15 and 17 above.	

31. The method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

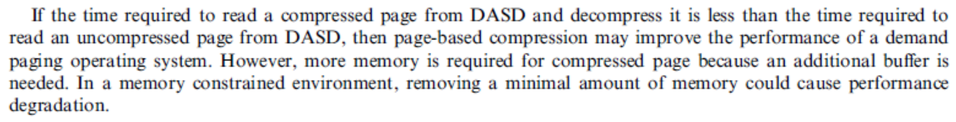
See Claims 1.1 and 1.2 above



1.1. Motivation

As microprocessors increase in speed many traditional compute-constrained tasks are transformed into Input/Output (I/O) bound task. Access to secondary memory or direct access storage devices (DASD) is quickly becoming the primary bottleneck for many computer Systems. Engineers continue to make great strides in processor speed, the processing power of the Intel X86 family of processors has increased 335 times between 1978 and 1993 (Wirth, 1995). Such amazing increase in performance has not been seen in DASD, however. As the performance gap widens, system performance will tend to become bounded by I/O performance. This is similar to Amdahl's Law: The small DASD-access component of a problem will limit the speed-up attainable by increasing processor speed. Thus, it is becoming much more important to understand the dynamics of DASD performance if we are to improve overall system performance.

Wynn Article, 1.1



If the time required to read a compressed page from DASD and decompress it is less than the time required to read an uncompressed page from DASD, then page-based compression may improve the performance of a demand paging operating system. However, more memory is required for compressed page because an additional buffer is needed. In a memory constrained environment, removing a minimal amount of memory could cause performance degradation.

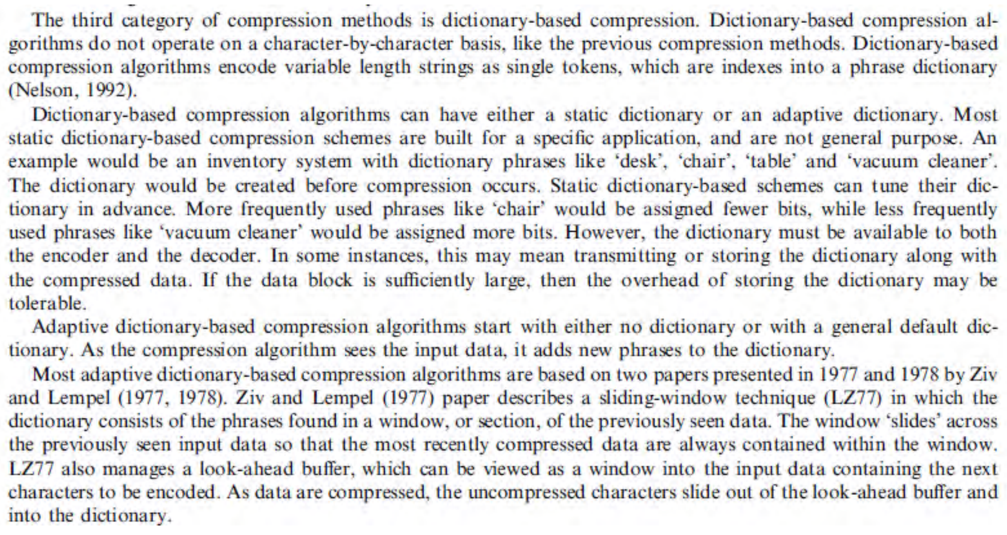
Wynn Article, 1.2

32. The method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claim 1.1 above

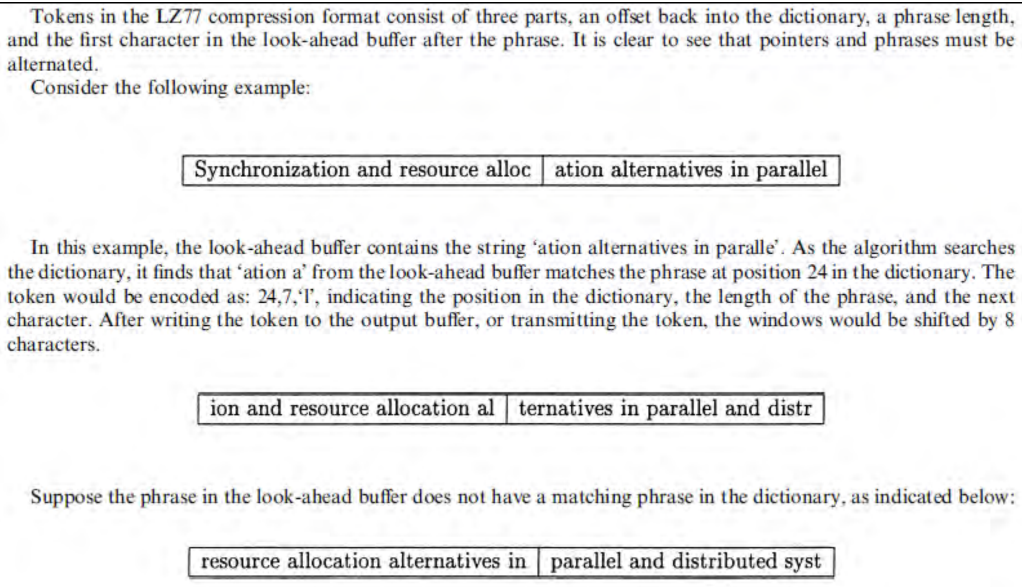

The third category of compression methods is dictionary-based compression. Dictionary-based compression algorithms do not operate on a character-by-character basis, like the previous compression methods. Dictionary-based compression algorithms encode variable length strings as single tokens, which are indexes into a phrase dictionary (Nelson, 1992).

Dictionary-based compression algorithms can have either a static dictionary or an adaptive dictionary. Most static dictionary-based compression schemes are built for a specific application, and are not general purpose. An example would be an inventory system with dictionary phrases like 'desk', 'chair', 'table' and 'vacuum cleaner'. The dictionary would be created before compression occurs. Static dictionary-based schemes can tune their dictionary in advance. More frequently used phrases like 'chair' would be assigned fewer bits, while less frequently used phrases like 'vacuum cleaner' would be assigned more bits. However, the dictionary must be available to both the encoder and the decoder. In some instances, this may mean transmitting or storing the dictionary along with the compressed data. If the data block is sufficiently large, then the overhead of storing the dictionary may be tolerable.

Adaptive dictionary-based compression algorithms start with either no dictionary or with a general default dictionary. As the compression algorithm sees the input data, it adds new phrases to the dictionary.

Most adaptive dictionary-based compression algorithms are based on two papers presented in 1977 and 1978 by Ziv and Lempel (1977, 1978). Ziv and Lempel (1977) paper describes a sliding-window technique (LZ77) in which the dictionary consists of the phrases found in a window, or section, of the previously seen data. The window 'slides' across the previously seen input data so that the most recently compressed data are always contained within the window. LZ77 also manages a look-ahead buffer, which can be viewed as a window into the input data containing the next characters to be encoded. As data are compressed, the uncompressed characters slide out of the look-ahead buffer and into the dictionary.

Wynn Article, 2.1



Tokens in the LZ77 compression format consist of three parts, an offset back into the dictionary, a phrase length, and the first character in the look-ahead buffer after the phrase. It is clear to see that pointers and phrases must be alternated.

Consider the following example:

Synchronization and resource alloc	ation alternatives in parallel

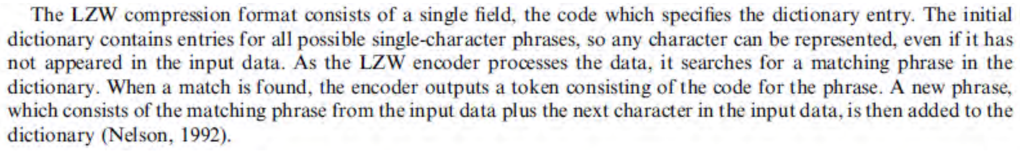
In this example, the look-ahead buffer contains the string 'ation alternatives in paralle'. As the algorithm searches the dictionary, it finds that 'ation a' from the look-ahead buffer matches the phrase at position 24 in the dictionary. The token would be encoded as: 24,7,'l', indicating the position in the dictionary, the length of the phrase, and the next character. After writing the token to the output buffer, or transmitting the token, the windows would be shifted by 8 characters.

ion and resource allocation al	ternatives in parallel and distr

Suppose the phrase in the look-ahead buffer does not have a matching phrase in the dictionary, as indicated below:

resource allocation alternatives in	parallel and distributed syst

Wynn Article, 2.1



The LZW compression format consists of a single field, the code which specifies the dictionary entry. The initial dictionary contains entries for all possible single-character phrases, so any character can be represented, even if it has not appeared in the input data. As the LZW encoder processes the data, it searches for a matching phrase in the dictionary. When a match is found, the encoder outputs a token consisting of the code for the phrase. A new phrase, which consists of the matching phrase from the input data plus the next character in the input data, is then added to the dictionary (Nelson, 1992).

Wynn Article, 2.1

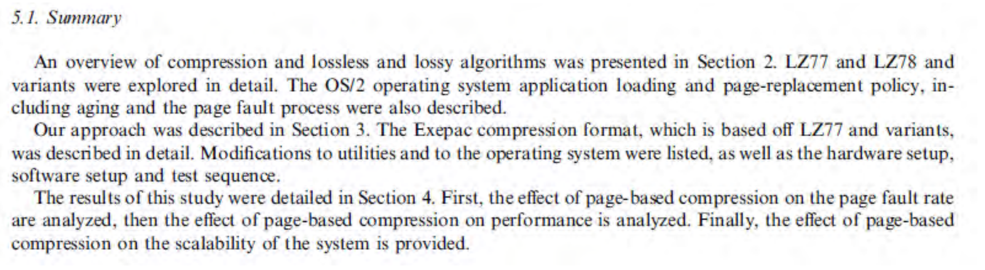
Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

33. The method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form.	Wynn, as evidenced by the example citations below, discloses "the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.1 and 32 above.



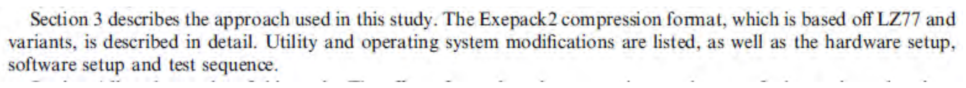
5.1. Summary

An overview of compression and lossless and lossy algorithms was presented in Section 2. LZ77 and LZ78 and variants were explored in detail. The OS/2 operating system application loading and page-replacement policy, including aging and the page fault process were also described.

Our approach was described in Section 3. The Exepac compression format, which is based off LZ77 and variants, was described in detail. Modifications to utilities and to the operating system were listed, as well as the hardware setup, software setup and test sequence.

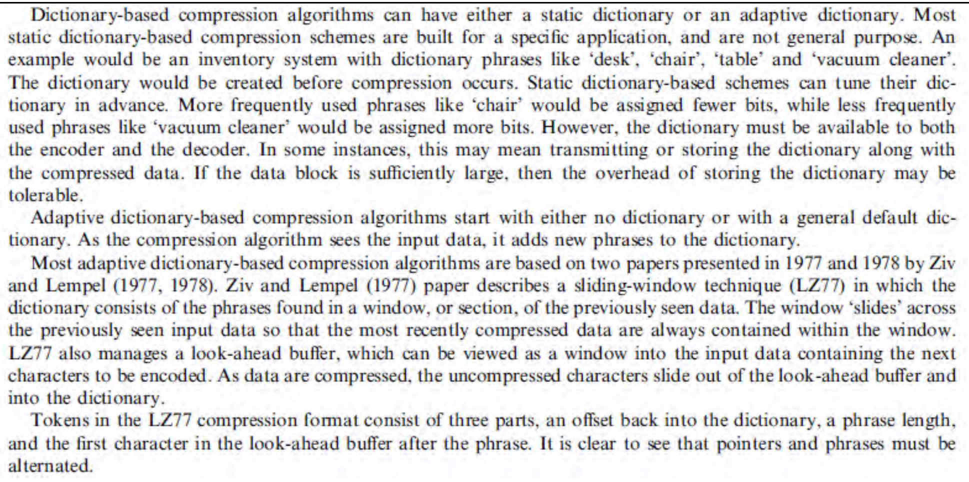
The results of this study were detailed in Section 4. First, the effect of page-based compression on the page fault rate are analyzed, then the effect of page-based compression on performance is analyzed. Finally, the effect of page-based compression on the scalability of the system is provided.

Wynn Article, 5.1



Section 3 describes the approach used in this study. The Exepack2 compression format, which is based off LZ77 and variants, is described in detail. Utility and operating system modifications are listed, as well as the hardware setup, software setup and test sequence.

Wynn Article, 1.2

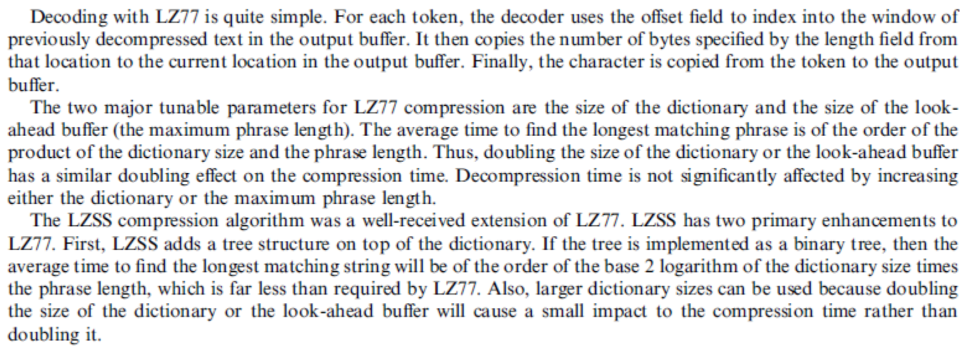

Dictionary-based compression algorithms can have either a static dictionary or an adaptive dictionary. Most static dictionary-based compression schemes are built for a specific application, and are not general purpose. An example would be an inventory system with dictionary phrases like 'desk', 'chair', 'table' and 'vacuum cleaner'. The dictionary would be created before compression occurs. Static dictionary-based schemes can tune their dictionary in advance. More frequently used phrases like 'chair' would be assigned fewer bits, while less frequently used phrases like 'vacuum cleaner' would be assigned more bits. However, the dictionary must be available to both the encoder and the decoder. In some instances, this may mean transmitting or storing the dictionary along with the compressed data. If the data block is sufficiently large, then the overhead of storing the dictionary may be tolerable.

Adaptive dictionary-based compression algorithms start with either no dictionary or with a general default dictionary. As the compression algorithm sees the input data, it adds new phrases to the dictionary.

Most adaptive dictionary-based compression algorithms are based on two papers presented in 1977 and 1978 by Ziv and Lempel (1977, 1978). Ziv and Lempel (1977) paper describes a sliding-window technique (LZ77) in which the dictionary consists of the phrases found in a window, or section, of the previously seen data. The window 'slides' across the previously seen input data so that the most recently compressed data are always contained within the window. LZ77 also manages a look-ahead buffer, which can be viewed as a window into the input data containing the next characters to be encoded. As data are compressed, the uncompressed characters slide out of the look-ahead buffer and into the dictionary.

Tokens in the LZ77 compression format consist of three parts, an offset back into the dictionary, a phrase length, and the first character in the look-ahead buffer after the phrase. It is clear to see that pointers and phrases must be alternated.

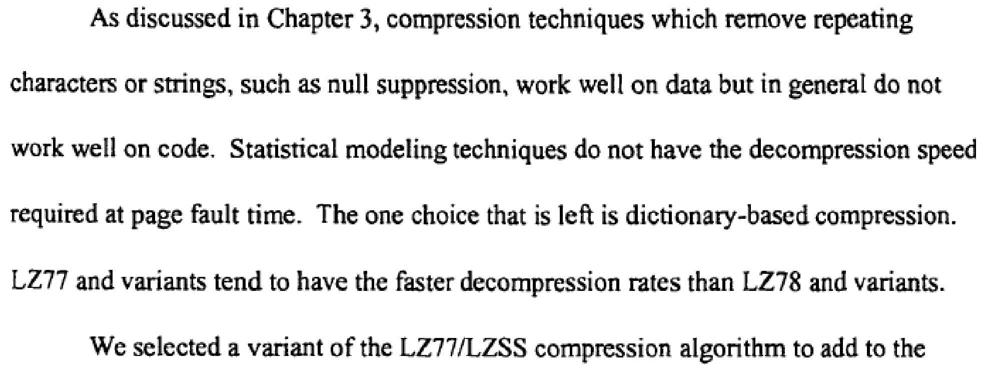
Wynn Article, 2.1


Decoding with LZ77 is quite simple. For each token, the decoder uses the offset field to index into the window of previously decompressed text in the output buffer. It then copies the number of bytes specified by the length field from that location to the current location in the output buffer. Finally, the character is copied from the token to the output buffer.

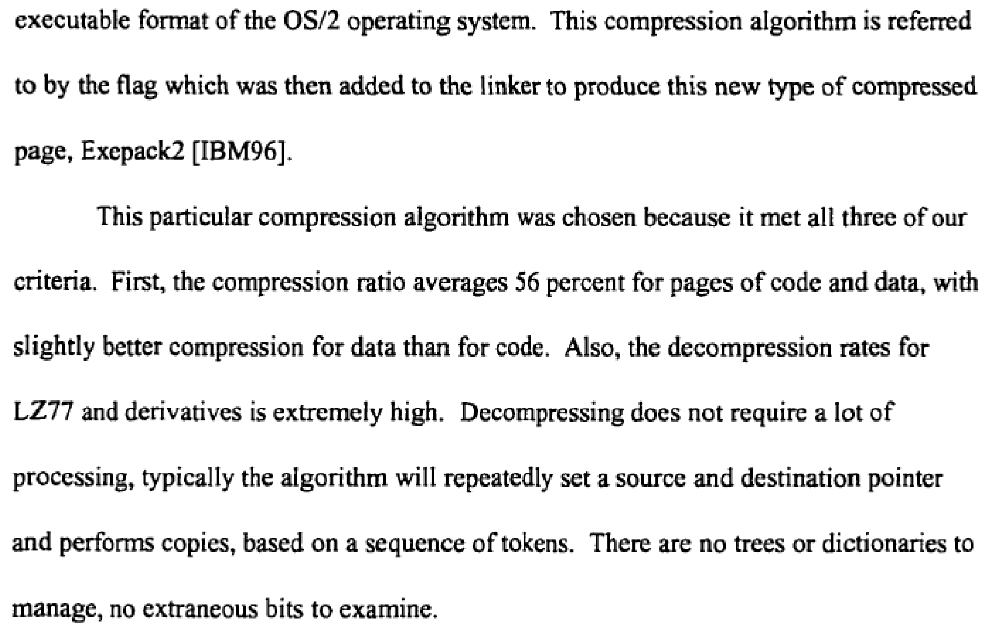
The two major tunable parameters for LZ77 compression are the size of the dictionary and the size of the look-ahead buffer (the maximum phrase length). The average time to find the longest matching phrase is of the order of the product of the dictionary size and the phrase length. Thus, doubling the size of the dictionary or the look-ahead buffer has a similar doubling effect on the compression time. Decompression time is not significantly affected by increasing either the dictionary or the maximum phrase length.

The LZSS compression algorithm was a well-received extension of LZ77. LZSS has two primary enhancements to LZ77. First, LZSS adds a tree structure on top of the dictionary. If the tree is implemented as a binary tree, then the average time to find the longest matching string will be of the order of the base 2 logarithm of the dictionary size times the phrase length, which is far less than required by LZ77. Also, larger dictionary sizes can be used because doubling the size of the dictionary or the look-ahead buffer will cause a small impact to the compression time rather than doubling it.

Wynn Article, 2.1


As discussed in Chapter 3, compression techniques which remove repeating characters or strings, such as null suppression, work well on data but in general do not work well on code. Statistical modeling techniques do not have the decompression speed required at page fault time. The one choice that is left is dictionary-based compression. LZ77 and variants tend to have the faster decompression rates than LZ78 and variants.

We selected a variant of the LZ77/LZSS compression algorithm to add to the


executable format of the OS/2 operating system. This compression algorithm is referred to by the flag which was then added to the linker to produce this new type of compressed page, Exepack2 [IBM96].

This particular compression algorithm was chosen because it met all three of our criteria. First, the compression ratio averages 56 percent for pages of code and data, with slightly better compression for data than for code. Also, the decompression rates for LZ77 and derivatives is extremely high. Decompressing does not require a lot of processing, typically the algorithm will repeatedly set a source and destination pointer and performs copies, based on a sequence of tokens. There are no trees or dictionaries to manage, no extraneous bits to examine.

Wynn Thesis, 5.2

35. The method of claim 5, wherein the compressed boot data represents a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

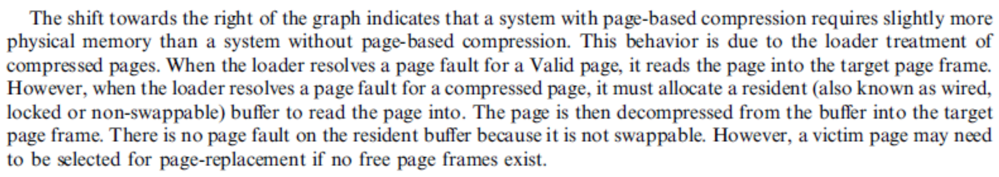
36. The method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system.	Wynn, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15 and 17 above.	

39. The method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory.	Wynn, as evidenced by the example citations below, discloses "the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory"

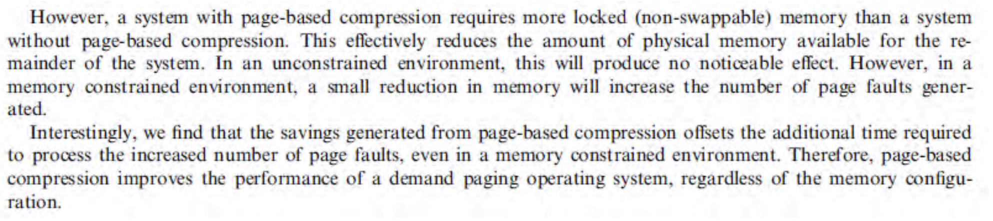
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.1 and 1.2 above


The shift towards the right of the graph indicates that a system with page-based compression requires slightly more physical memory than a system without page-based compression. This behavior is due to the loader treatment of compressed pages. When the loader resolves a page fault for a Valid page, it reads the page into the target page frame. However, when the loader resolves a page fault for a compressed page, it must allocate a resident (also known as wired, locked or non-swappable) buffer to read the page into. The page is then decompressed from the buffer into the target page frame. There is no page fault on the resident buffer because it is not swappable. However, a victim page may need to be selected for page-replacement if no free page frames exist.

Wynn Article, 4.2


However, a system with page-based compression requires more locked (non-swappable) memory than a system without page-based compression. This effectively reduces the amount of physical memory available for the remainder of the system. In an unconstrained environment, this will produce no noticeable effect. However, in a memory constrained environment, a small reduction in memory will increase the number of page faults generated.

Interestingly, we find that the savings generated from page-based compression offsets the additional time required to process the increased number of page faults, even in a memory constrained environment. Therefore, page-based compression improves the performance of a demand paging operating system, regardless of the memory configuration.

Wynn Article, 5.2

40. The method of claim 36, wherein the operating system comprises: a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 36, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 36, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

43. The method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access.	Wynn, as evidenced by the example citations below, discloses "the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 31 above.	

44. The method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Wynn, as evidenced by the example citations below, discloses "the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 32 above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

45. The method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Wynn, as evidenced by the example citations below, discloses "the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 32 and 33 above.	

47. The system of claim 6, wherein the boot data in the compressed form represents a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

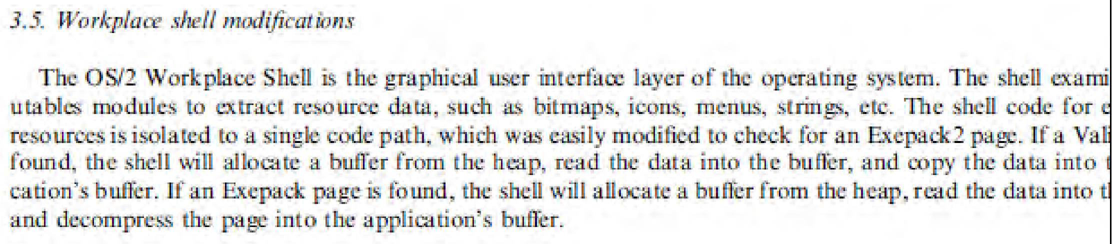
48. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system.	Wynn, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15, 17 and 24 above.	

49. The system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form.	Wynn, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

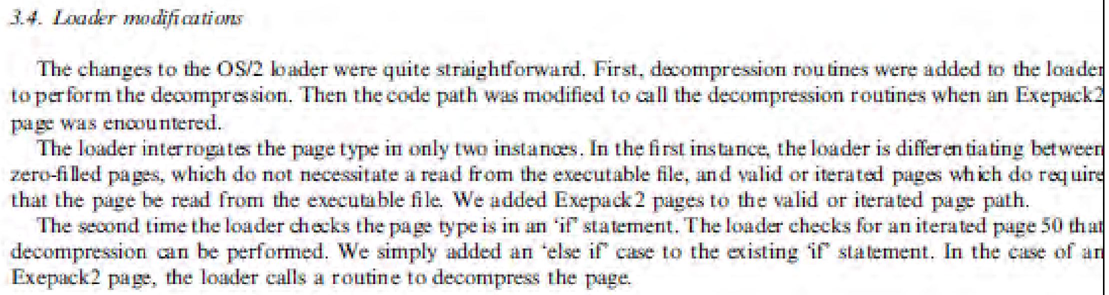
See Claim 10 above.



3.5. Workplace shell modifications

The OS/2 Workplace Shell is the graphical user interface layer of the operating system. The shell exami
utables modules to extract resource data, such as bitmaps, icons, menus, strings, etc. The shell code for e
resources is isolated to a single code path, which was easily modified to check for an Exepack2 page. If a Val
found, the shell will allocate a buffer from the heap, read the data into the buffer, and copy the data into t
cation's buffer. If an Exepack page is found, the shell will allocate a buffer from the heap, read the data into t
and decompress the page into the application's buffer.

Wynn Article, 3.5



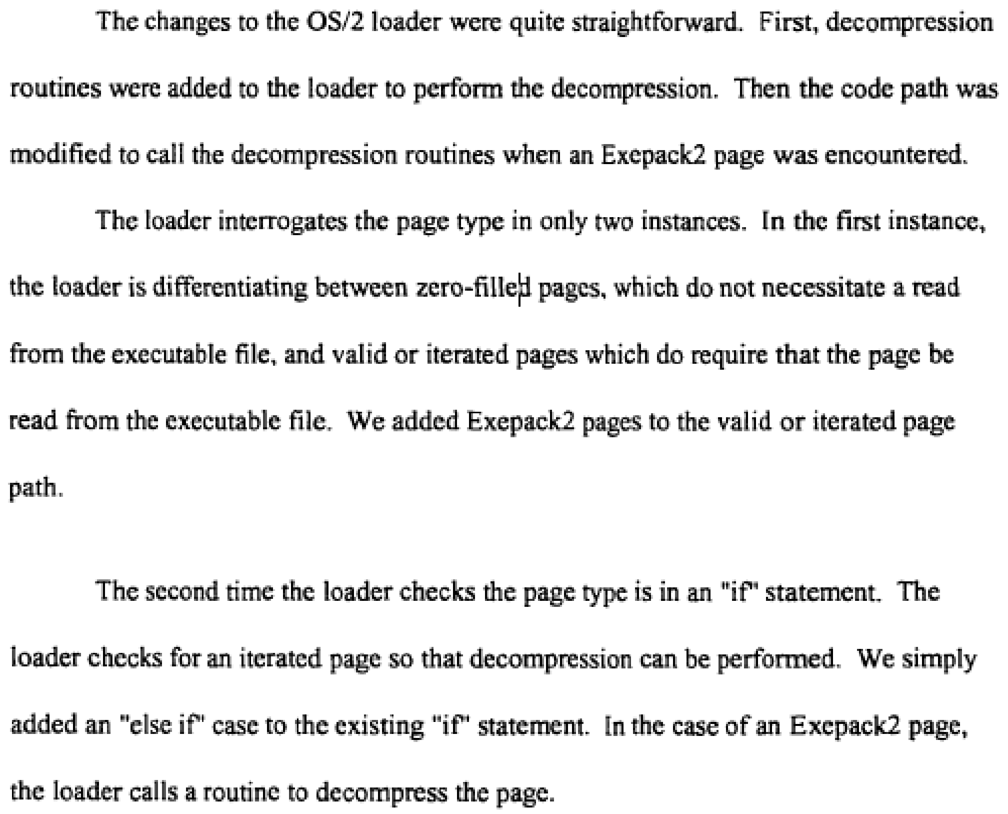
3.4. Loader modifications

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader
to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2
page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between
zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require
that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that
decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an
Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article 3.4



The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

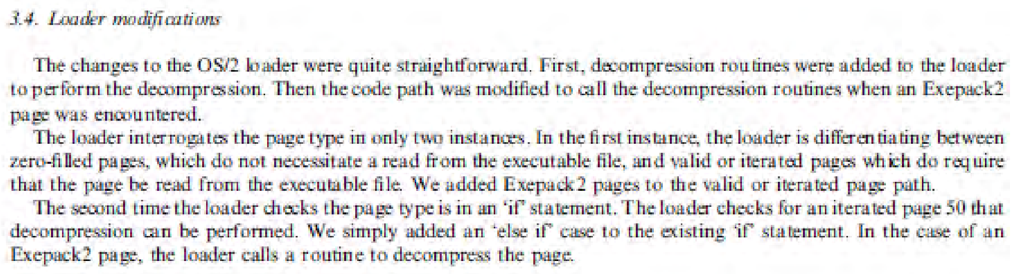
The second time the loader checks the page type is in an "if" statement. The loader checks for an iterated page so that decompression can be performed. We simply added an "else if" case to the existing "if" statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Thesis, 5.5

50. The system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form.	Wynn, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:



3.4. Loader modifications

The changes to the OS/2 loader were quite straightforward. First, decompression routines were added to the loader to perform the decompression. Then the code path was modified to call the decompression routines when an Exepack2 page was encountered.

The loader interrogates the page type in only two instances. In the first instance, the loader is differentiating between zero-filled pages, which do not necessitate a read from the executable file, and valid or iterated pages which do require that the page be read from the executable file. We added Exepack2 pages to the valid or iterated page path.

The second time the loader checks the page type is in an 'if' statement. The loader checks for an iterated page 50 that decompression can be performed. We simply added an 'else if' case to the existing 'if' statement. In the case of an Exepack2 page, the loader calls a routine to decompress the page.

Wynn Article 3.4

51. The system of claim 6, wherein the first memory comprises: a physical memory.	Wynn, as evidenced by the example citations below, discloses "the system of claim 6, wherein the first memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the first memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 27 and 39 above.	

52. The system of claim 6, wherein the boot data the compressed form comprises: a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

53. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program.	Wynn, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15, 17 and 24 above.	

59. The method of claim 8, wherein the operating system in the compressed form represents a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

60. The method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system.	Wynn, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15, 17 and 24 above.	

63. The method of claim 8, wherein the second memory comprises: a physical memory.	Wynn, as evidenced by the example citations below, discloses "the method of claim 8, wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 27 and 38 above.	

64. The method of claim 8, wherein the operating system comprises: a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

65. The method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program.	Wynn, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15, 17 and 24 above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

67. The method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access.	Wynn, as evidenced by the example citations below, discloses "the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 31 above.	

71. The method of claim 11, wherein the boot data in the compressed form represents a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

72. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Wynn, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15, 17 and 24 above.	

75. The method of claim 11, wherein the memory comprises: a physical memory.	Wynn, as evidenced by the example citations below, discloses “the method of claim 11, wherein the memory comprises: a physical memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Wynn discloses this limitation: *See* Claim 27 above.	

76. The method of claim 11, wherein the operating system comprises: a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 11, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

77. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program.	Wynn, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15, 17 and 24 above.	

79. The method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access.	Wynn, as evidenced by the example citations below, discloses "the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 31 above.	

80. The method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form.	Wynn, as evidenced by the example citations below, discloses "the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 32, 33 and 49 above.	

81. The method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form.	Wynn, as evidenced by the example citations below, discloses "the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 32, 33 and 49 above.	

83. The method of claim 13, wherein the boot data in the compressed form represents a plurality of files.	Wynn, as evidenced by the example citations below, discloses “the method of claim 13, wherein the boot data in the compressed form represents a plurality of files”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

84. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Wynn, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 15, 17 and 24 above.	

87. The method of claim 13, wherein the memory comprises: a physical memory.	Wynn, as evidenced by the example citations below, discloses "the method of claim 13, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 27 above.	

Appendix C37
Invalidity of U.S. Patent 8,880,862 based on Wynn

88. The method of claim 13, wherein the operating system comprises: a plurality of files.	Wynn, as evidenced by the example citations below, discloses "the method of claim 13, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 16 and 23 above.	

89. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program.	Wynn, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 15, 17 and 24 above.

91. The method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access.	Wynn, as evidenced by the example citations below, discloses "the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 31 above.	

92. The method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Wynn, as evidenced by the example citations below, discloses "the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 32, 33 and 49 above.	

93. The method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Wynn, as evidenced by the example citations below, discloses "the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claim 32, 33, and 49 above.	

97.1 The method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list,	Wynn, as evidenced by the example citations below, discloses "the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 9.1-9.3 and 19 above.

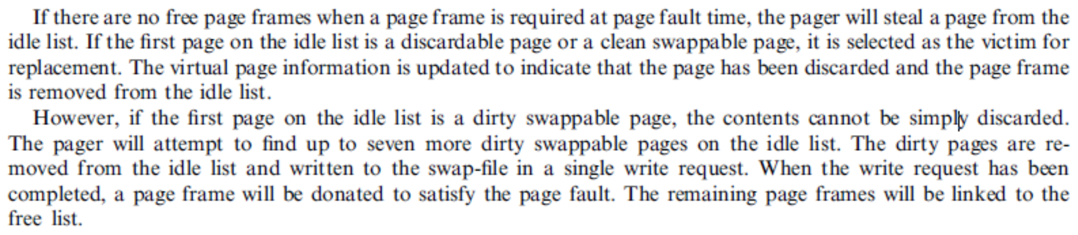
97.2 and wherein the updating comprises: associating the additional compressed boot data with the boot data list.	Wynn, as evidenced by the example citations below, discloses "and wherein the updating comprises: associating the additional compressed boot data with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, and wherein the updating comprises: associating the additional compressed boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.4.1, 2, 4, 5.6, and 8.6 above.	

98. The method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list.	Wynn, as evidenced by the example citations below, discloses “the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list”

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.4.1, 5.6, and 8.6 above.



If there are no free page frames when a page frame is required at page fault time, the pager will steal a page from the idle list. If the first page on the idle list is a discardable page or a clean swappable page, it is selected as the victim for replacement. The virtual page information is updated to indicate that the page has been discarded and the page frame is removed from the idle list.

However, if the first page on the idle list is a dirty swappable page, the contents cannot be simply discarded. The pager will attempt to find up to seven more dirty swappable pages on the idle list. The dirty pages are removed from the idle list and written to the swap-file in a single write request. When the write request has been completed, a page frame will be donated to satisfy the page fault. The remaining page frames will be linked to the free list.

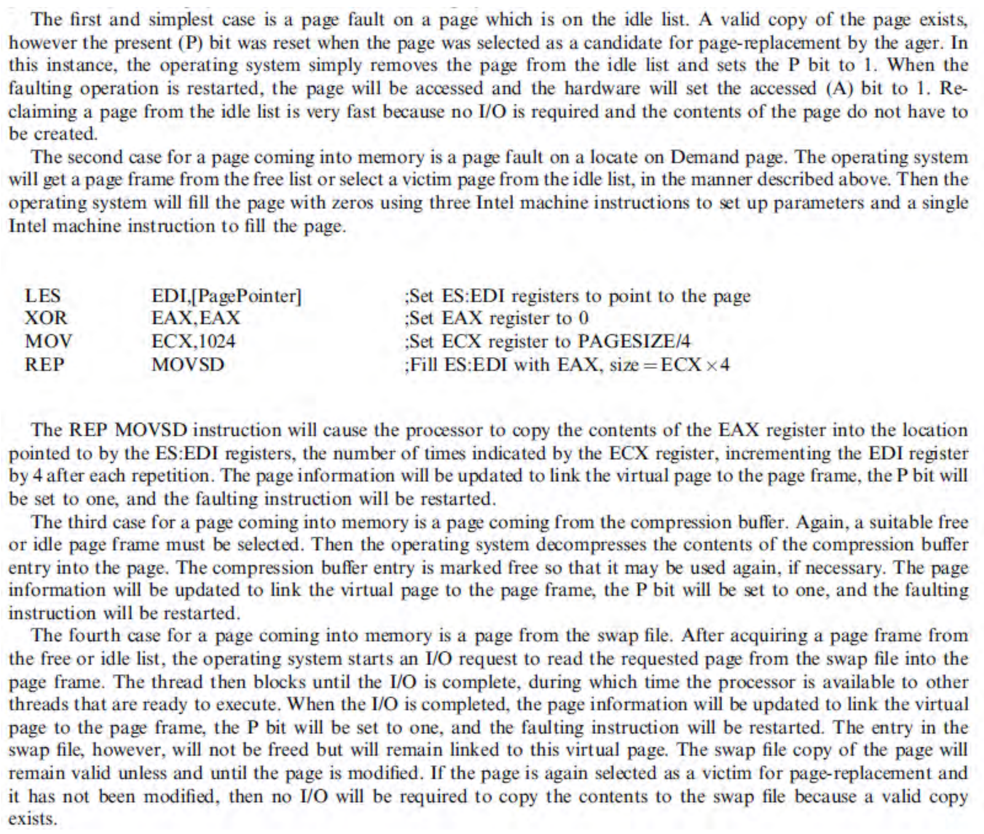
Wynn Article, 3.6

107. The method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.	Wynn, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 1.4.1, 5.6, and 8.6 above.



The first and simplest case is a page fault on a page which is on the idle list. A valid copy of the page exists, however the present (P) bit was reset when the page was selected as a candidate for page-replacement by the ager. In this instance, the operating system simply removes the page from the idle list and sets the P bit to 1. When the faulting operation is restarted, the page will be accessed and the hardware will set the accessed (A) bit to 1. Reclaiming a page from the idle list is very fast because no I/O is required and the contents of the page do not have to be created.

The second case for a page coming into memory is a page fault on a locate on Demand page. The operating system will get a page frame from the free list or select a victim page from the idle list, in the manner described above. Then the operating system will fill the page with zeros using three Intel machine instructions to set up parameters and a single Intel machine instruction to fill the page.

```
LES        EDI,[PagePointer]        ;Set ES:EDI registers to point to the page
XOR        EAX,EAX                  ;Set EAX register to 0
MOV        ECX,1024                 ;Set ECX register to PAGESIZE/4
REP        MOVSD                    ;Fill ES:EDI with EAX, size = ECX × 4
```

The REP MOVSD instruction will cause the processor to copy the contents of the EAX register into the location pointed to by the ES:EDI registers, the number of times indicated by the ECX register, incrementing the EDI register by 4 after each repetition. The page information will be updated to link the virtual page to the page frame, the P bit will be set to one, and the faulting instruction will be restarted.

The third case for a page coming into memory is a page coming from the compression buffer. Again, a suitable free or idle page frame must be selected. Then the operating system decompresses the contents of the compression buffer entry into the page. The compression buffer entry is marked free so that it may be used again, if necessary. The page information will be updated to link the virtual page to the page frame, the P bit will be set to one, and the faulting instruction will be restarted.

The fourth case for a page coming into memory is a page from the swap file. After acquiring a page frame from the free or idle list, the operating system starts an I/O request to read the requested page from the swap file into the page frame. The thread then blocks until the I/O is complete, during which time the processor is available to other threads that are ready to execute. When the I/O is completed, the page information will be updated to link the virtual page to the page frame, the P bit will be set to one, and the faulting instruction will be restarted. The entry in the swap file, however, will not be freed but will remain linked to this virtual page. The swap file copy of the page will remain valid unless and until the page is modified. If the page is again selected as a victim for page-replacement and it has not been modified, then no I/O will be required to copy the contents to the swap file because a valid copy exists.

Wynn Article, 3.6

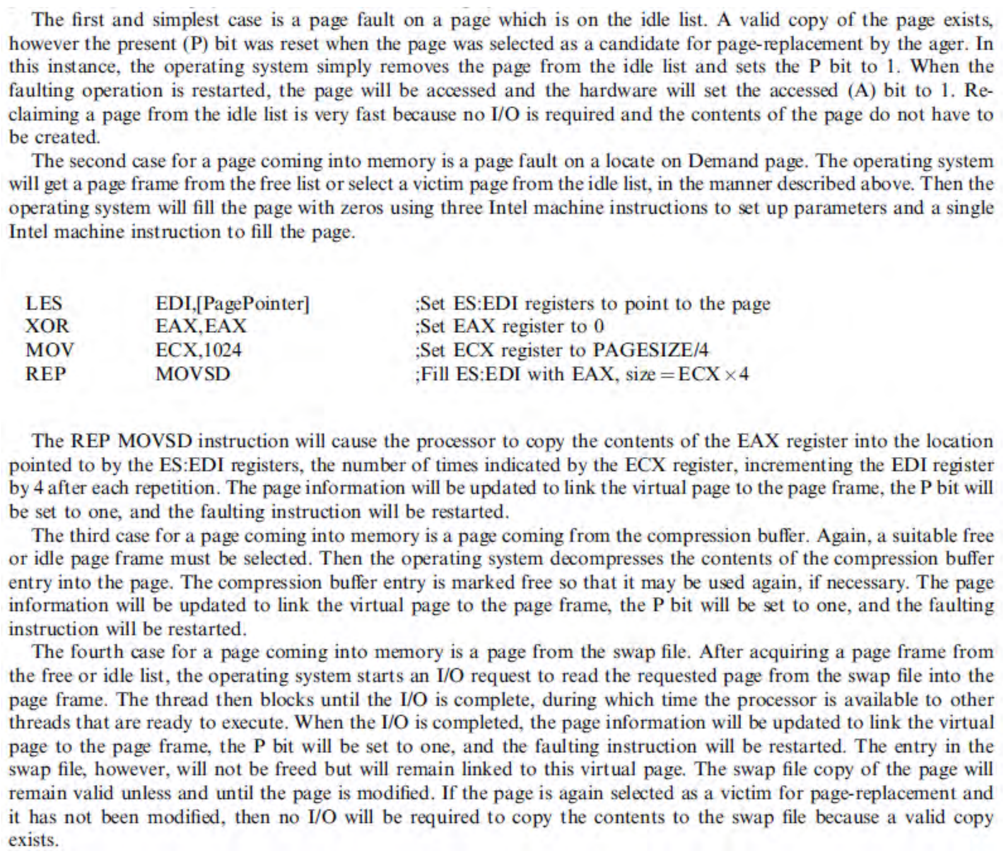
108. The method of claim 2, further comprising: compressing at least a portion of the additional boot data.	Wynn, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: compressing at least a portion of the additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: compressing at least a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Wynn discloses this limitation: *See* Claims 1.1, 5, 9.1 and 8 above.	

109. The method of claim 108, further comprising: storing the compressed additional boot data.	Wynn, as evidenced by the example citations below, discloses "the method of claim 108, further comprising: storing the compressed additional boot data"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 108, further comprising: storing the compressed additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Wynn discloses this limitation:

See Claims 5.1, 8.1 and 9.1 above.



The first and simplest case is a page fault on a page which is on the idle list. A valid copy of the page exists, however the present (P) bit was reset when the page was selected as a candidate for page-replacement by the ager. In this instance, the operating system simply removes the page from the idle list and sets the P bit to 1. When the faulting operation is restarted, the page will be accessed and the hardware will set the accessed (A) bit to 1. Reclaiming a page from the idle list is very fast because no I/O is required and the contents of the page do not have to be created.

The second case for a page coming into memory is a page fault on a locate on Demand page. The operating system will get a page frame from the free list or select a victim page from the idle list, in the manner described above. Then the operating system will fill the page with zeros using three Intel machine instructions to set up parameters and a single Intel machine instruction to fill the page.

```
LES      EDI,[PagePointer]      ;Set ES:EDI registers to point to the page
XOR      EAX,EAX                ;Set EAX register to 0
MOV      ECX,1024               ;Set ECX register to PAGESIZE/4
REP      MOVSD                  ;Fill ES:EDI with EAX, size=ECX×4
```

The REP MOVSD instruction will cause the processor to copy the contents of the EAX register into the location pointed to by the ES:EDI registers, the number of times indicated by the ECX register, incrementing the EDI register by 4 after each repetition. The page information will be updated to link the virtual page to the page frame, the P bit will be set to one, and the faulting instruction will be restarted.

The third case for a page coming into memory is a page coming from the compression buffer. Again, a suitable free or idle page frame must be selected. Then the operating system decompresses the contents of the compression buffer entry into the page. The compression buffer entry is marked free so that it may be used again, if necessary. The page information will be updated to link the virtual page to the page frame, the P bit will be set to one, and the faulting instruction will be restarted.

The fourth case for a page coming into memory is a page from the swap file. After acquiring a page frame from the free or idle list, the operating system starts an I/O request to read the requested page from the swap file into the page frame. The thread then blocks until the I/O is complete, during which time the processor is available to other threads that are ready to execute. When the I/O is completed, the page information will be updated to link the virtual page to the page frame, the P bit will be set to one, and the faulting instruction will be restarted. The entry in the swap file, however, will not be freed but will remain linked to this virtual page. The swap file copy of the page will remain valid unless and until the page is modified. If the page is again selected as a victim for page-replacement and it has not been modified, then no I/O will be required to copy the contents to the swap file because a valid copy exists.

Wynn Article, 3.6

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

Red Hat Linux 5.0, The Official Red Hat Linux Installation Guide (1995) ("Linux Redhat") and M. Beck, et. al, "Linux Kernel Internals" Addison Wesley Longman (1996) ("Beck") (collectively, "Linux Kernel"), alone or in combination, invalidate claims 1-6, 8-11, 13-17, 19, 23-24, 27-29, 31-33, 35-36, 39-40, 43-45, 47-53, 59-60, 63-65, 67, 71-72, 75-77, 79-81, 83-84, 87-89, 91-93, 97-98, and 107-109 of United States Patent No. 8,880,862 ("the '862 Patent") pursuant to 35 U.S.C. § 102 and/or 35 U.S.C. § 103 either alone or in combination with other prior art references, and/or in combination with the knowledge of a person of ordinary skill.

The analysis provided in this chart may in some instances uses Realtime's proposed (or implied) claim constructions, and Apple reserves all rights to challenge these proposed (or implied) constructions. To the extent any of the charted prior art should fail to disclose an element of any claims of the '862 Patent, Apple reserves the right to rely upon the knowledge of one skilled in the art, or any other disclosed prior art, alone or in combination, whether produced by Apple or by Realtime, to show the element and thereby invalidate those claims. Citations given in the chart below are merely representative of the respective elements and are not meant to be exhaustive.

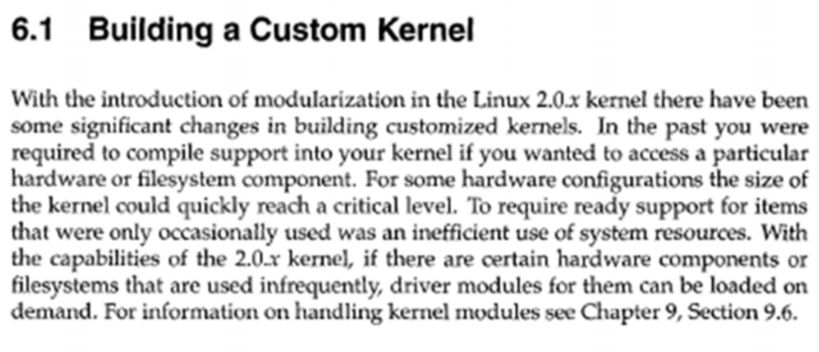
Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

1 (Preamble) A method for providing accelerated loading of an operating system in a computer system, the method comprising:	Linux Kernel, as evidenced by the exemplary citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, the method comprising:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accelerated loading of an operating system in a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

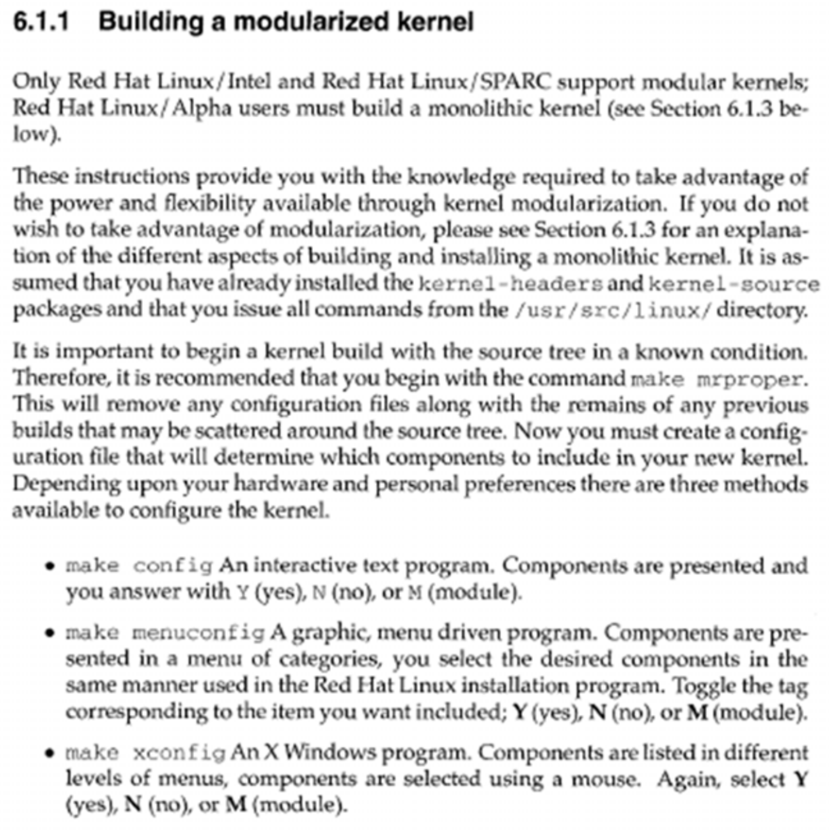
Linux Kernel discloses this limitation:

Linux Kernel discloses this limitation:


6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

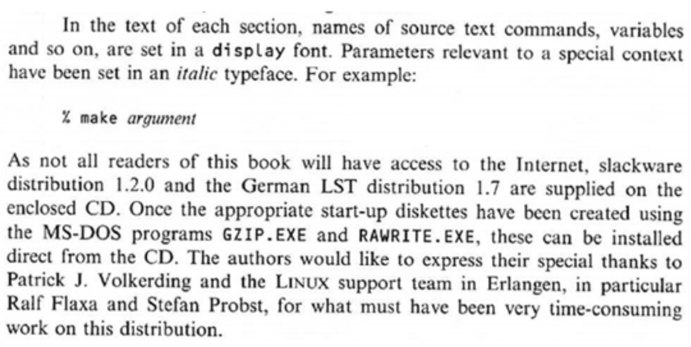
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1

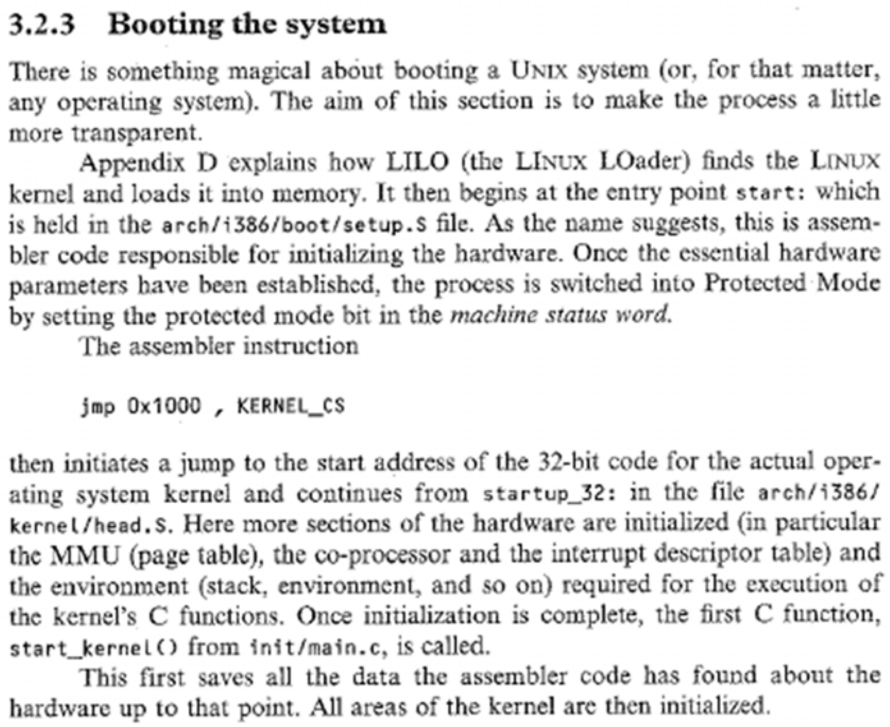


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface



3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word.*

The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

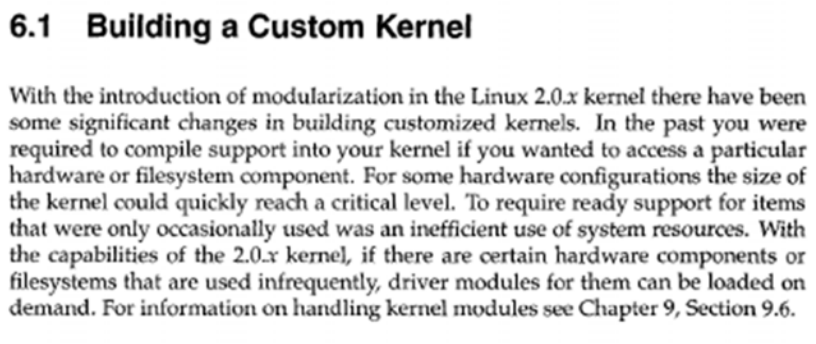
Beck, at 3.2.3

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

1.1 loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;	Linux Kernel, as evidenced by the example citations below, discloses "loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading a portion of boot data in a compressed form that is associated with a portion of a boot data list for booting the computer system into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

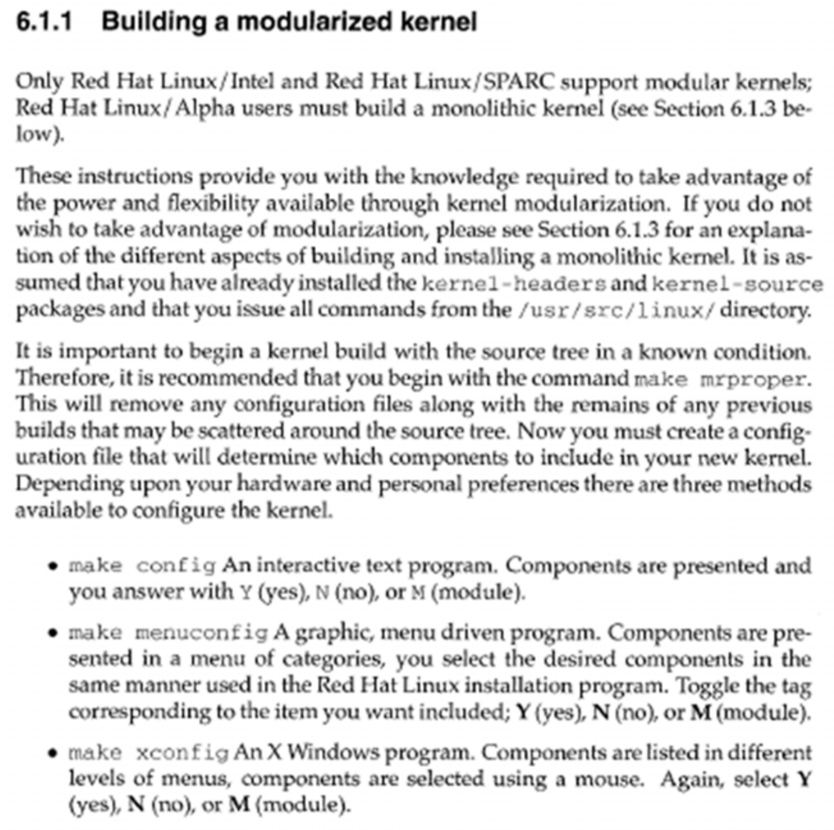
Linux Kernel discloses this limitation:



6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

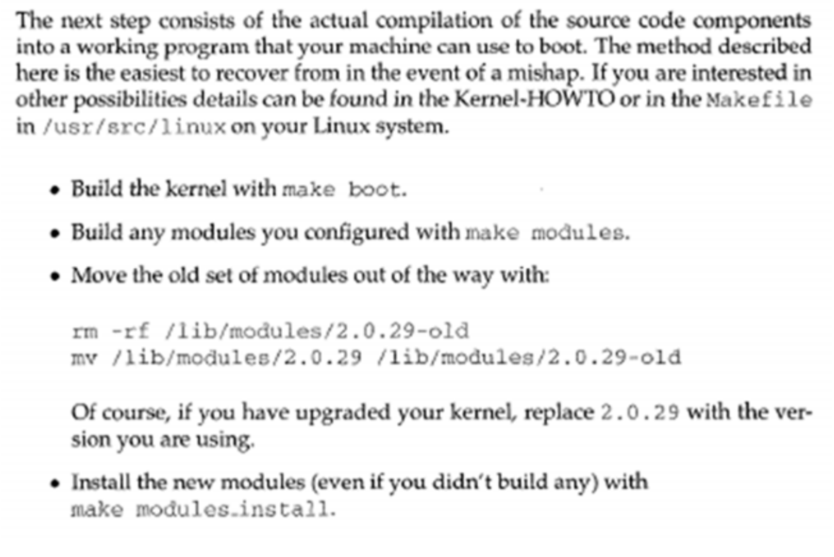
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1



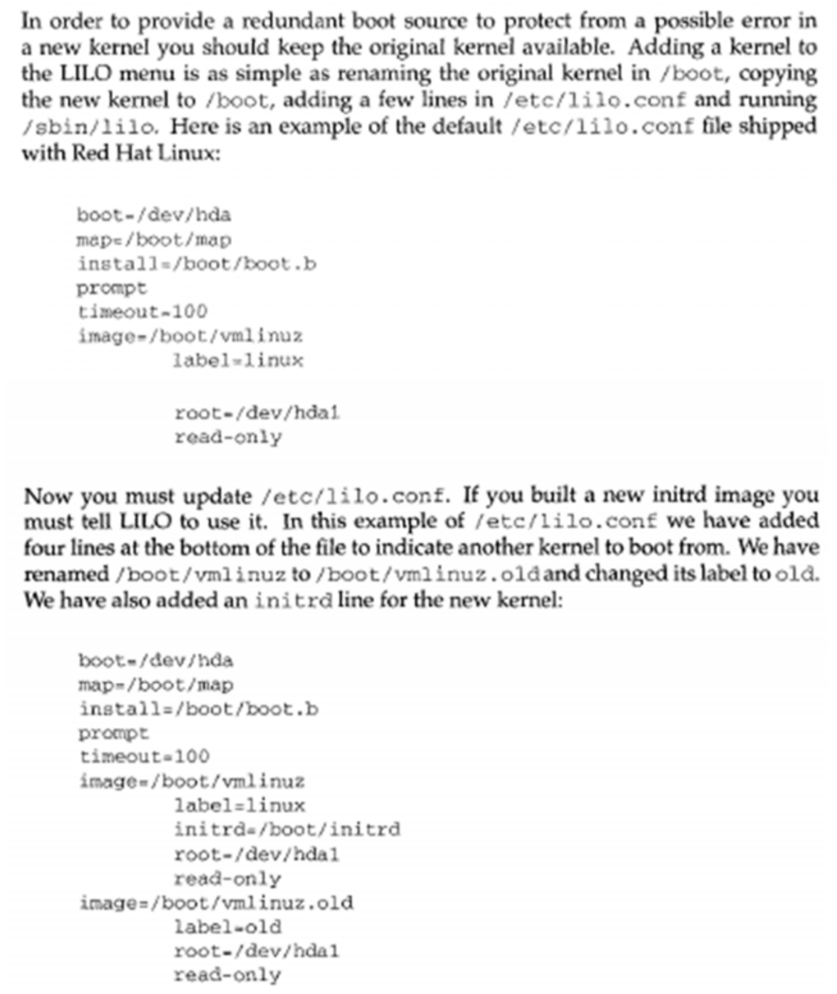
The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

```
rm -rf /lib/modules/2.0.29-old
mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1



In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in /boot, copying the new kernel to /boot, adding a few lines in /etc/lilo.conf and running /sbin/lilo. Here is an example of the default /etc/lilo.conf file shipped with Red Hat Linux:

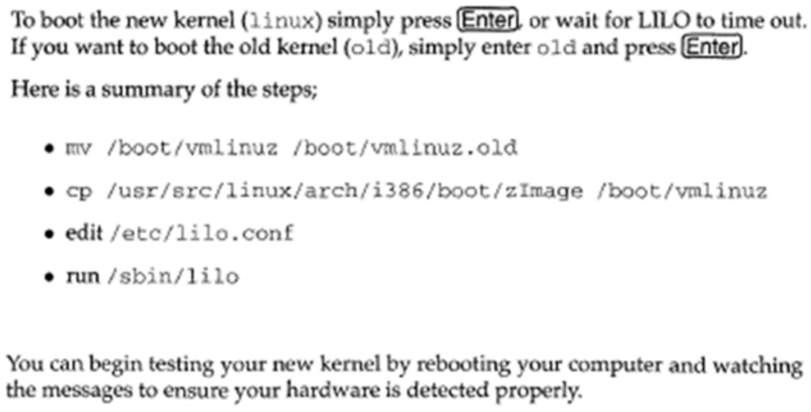
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update /etc/lilo.conf. If you built a new initrd image you must tell LILO to use it. In this example of /etc/lilo.conf we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed /boot/vmlinuz to /boot/vmlinuz.old and changed its label to old. We have also added an initrd line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1



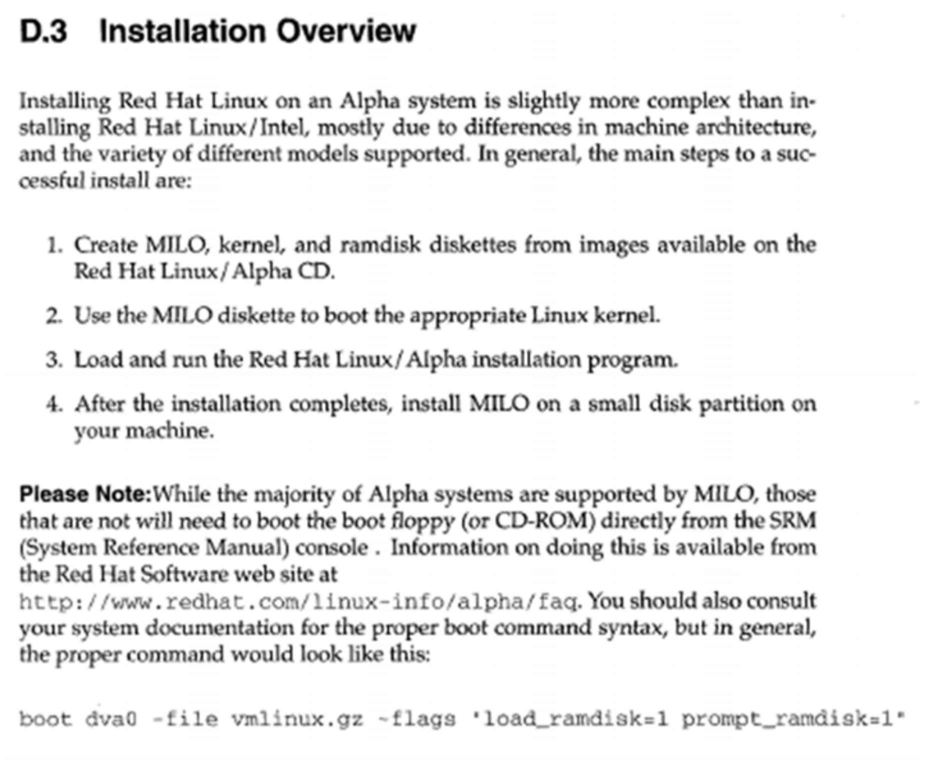
To boot the new kernel (linux) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (old), simply enter old and press Enter.

Here is a summary of the steps;

- mv /boot/vmlinuz /boot/vmlinuz.old
- cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz
- edit /etc/lilo.conf
- run /sbin/lilo

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1



D.3 Installation Overview

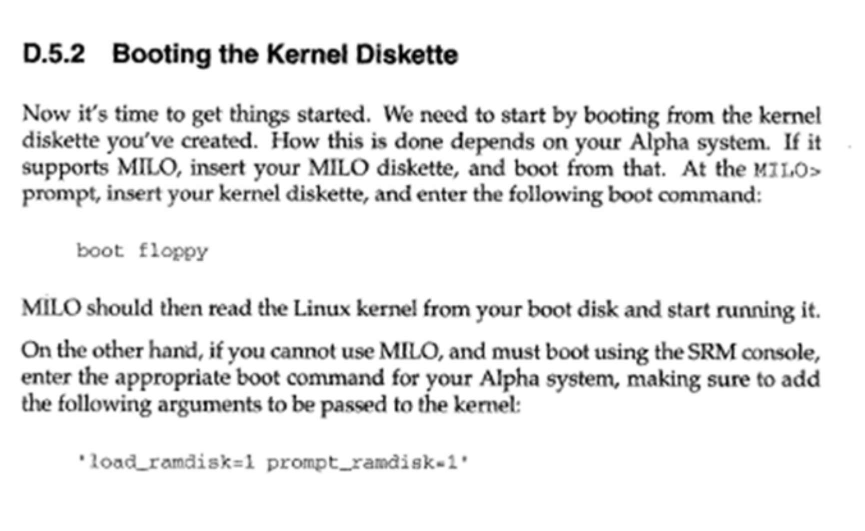
Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3



D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

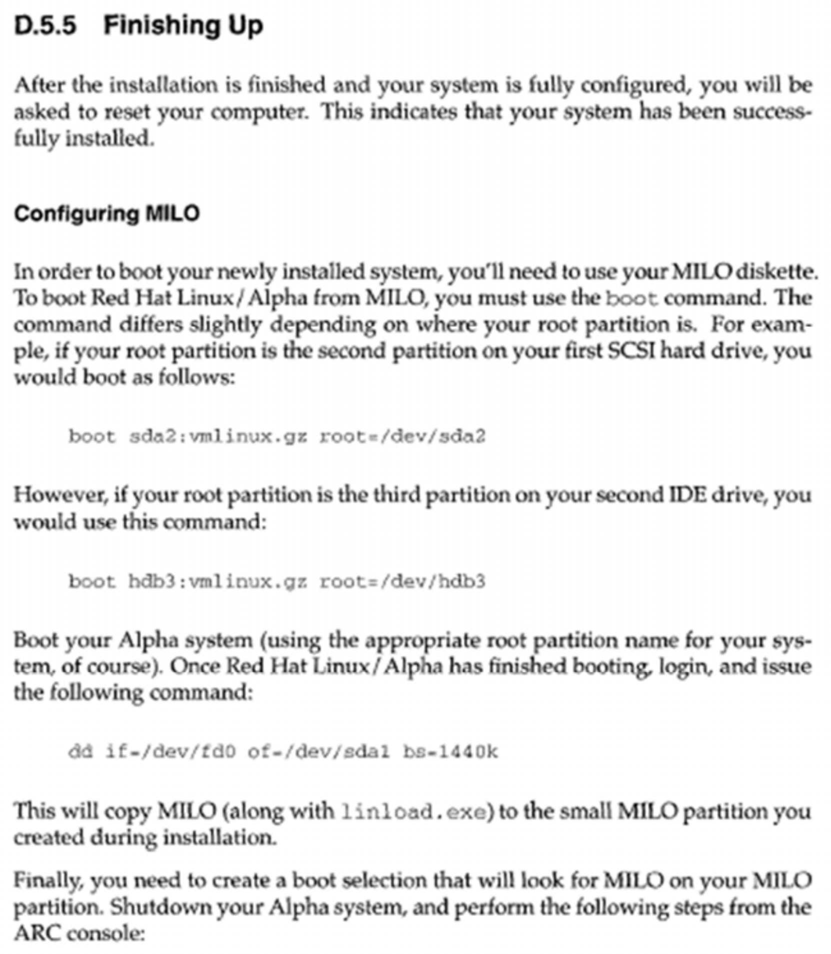
```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2



D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux / Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

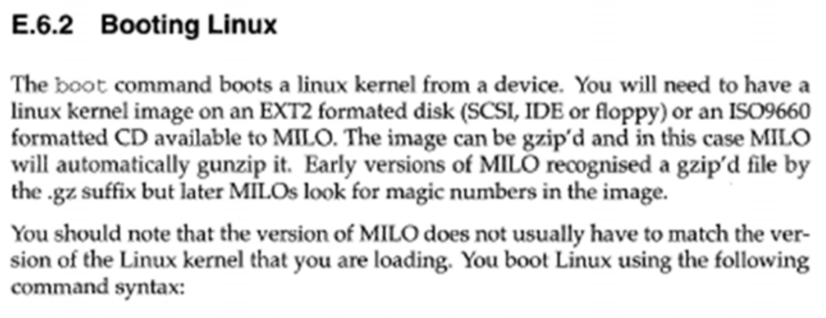
Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux / Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

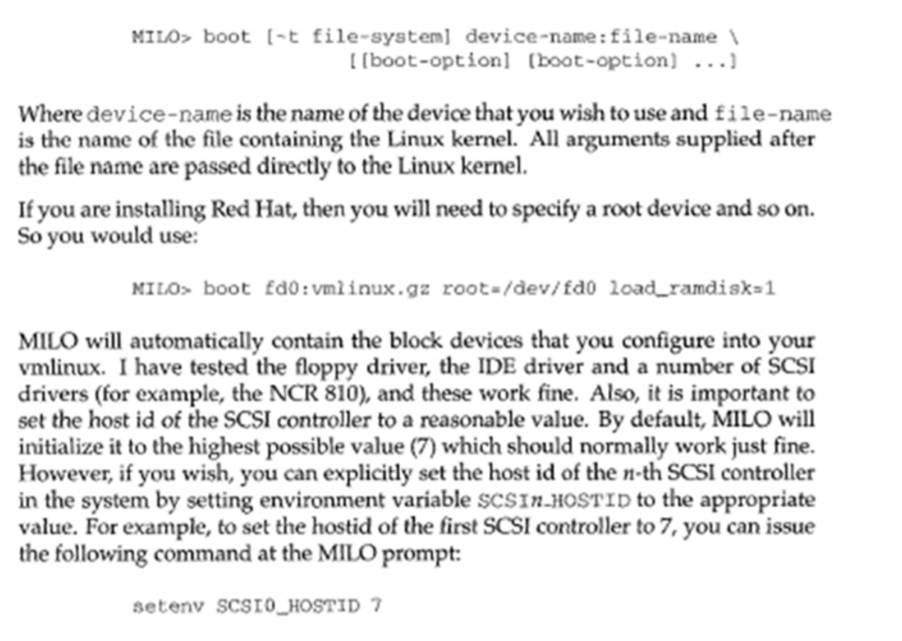
Linux Redhat, at Appendix D5.5



E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:



```
MILO> boot [-t file-system] device-name:file-name \
                  [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

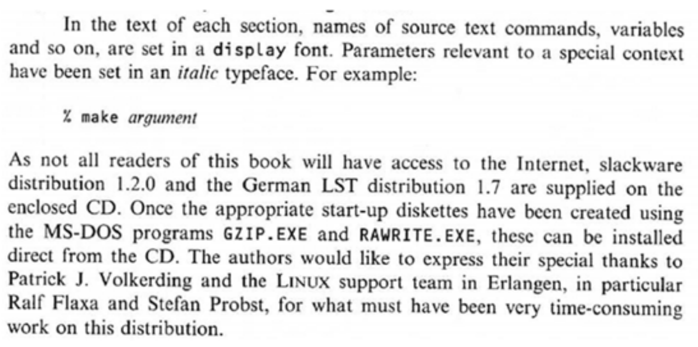
If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

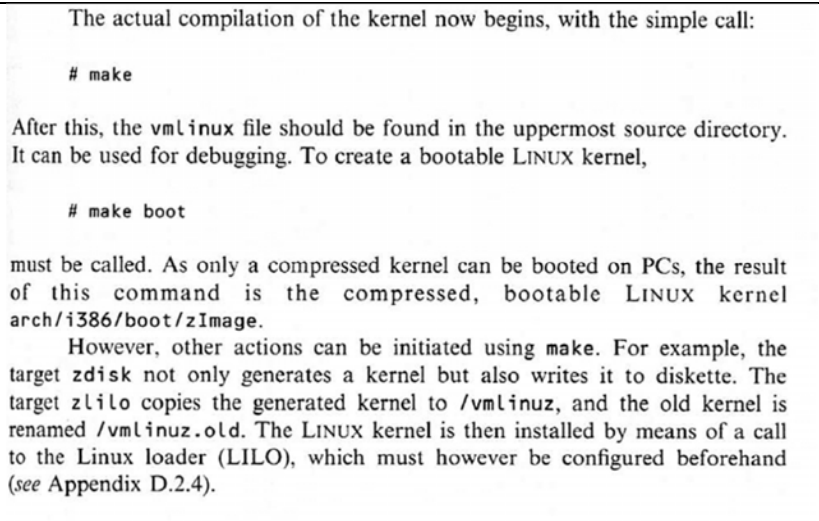


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface



The actual compilation of the kernel now begins, with the simple call:

```
# make
```

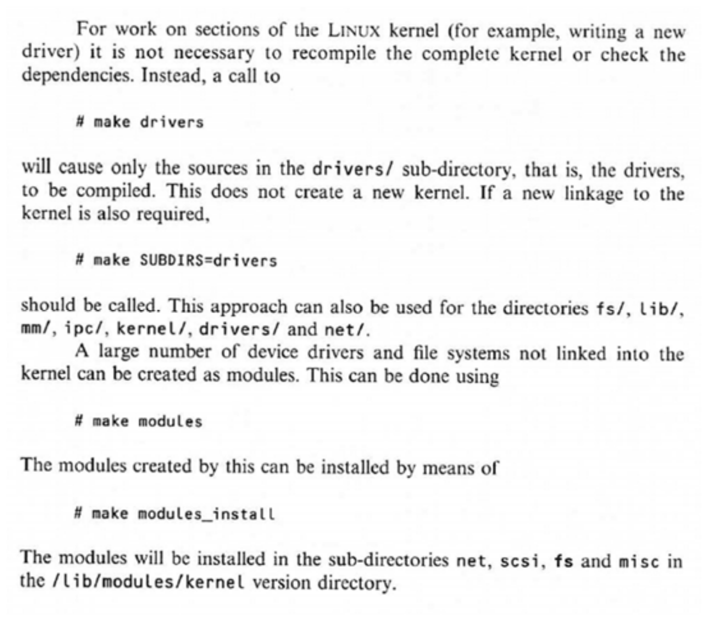
After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2



For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

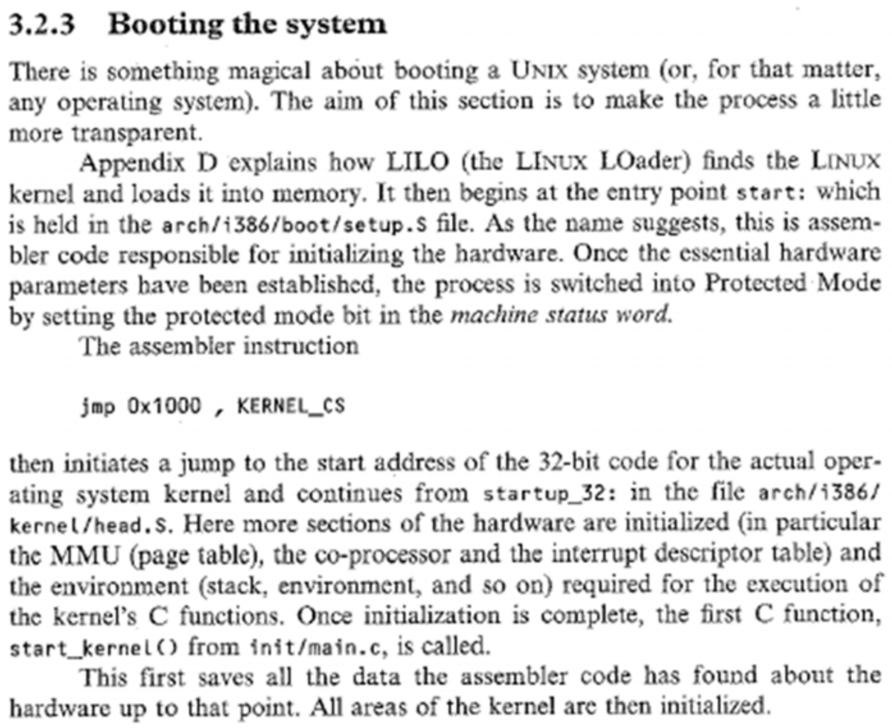
```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2



3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word*.

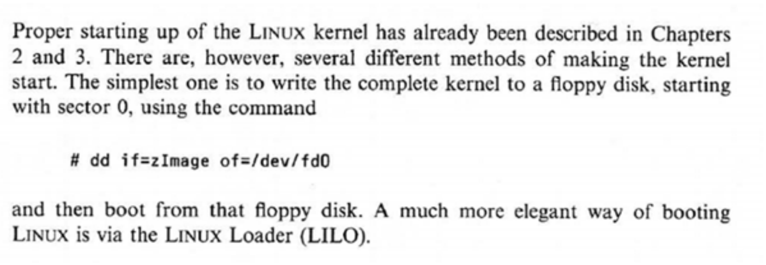
The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3



Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D

In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

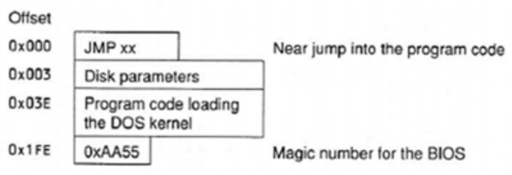

Offset
0x000
JMP xx
Near jump into the program code
0x003
Disk parameters
0x03E
Program code loading the DOS kernel
0x1FE
0xAA55
Magic number for the BIOS

Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

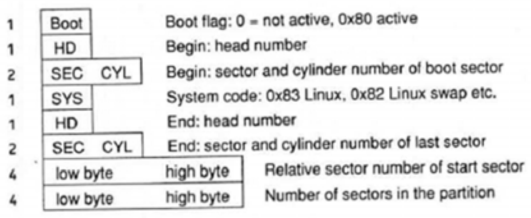

1
Boot
Boot flag: 0 = not active, 0x80 active
1
HD
Begin: head number
2
SEC CYL
Begin: sector and cylinder number of boot sector
1
SYS
System code: 0x83 Linux, 0x82 Linux swap etc.
1
HD
End: head number
2
SEC CYL
End: sector and cylinder number of last sector
4
low byte high byte
Relative sector number of start sector
4
low byte high byte
Number of sectors in the partition

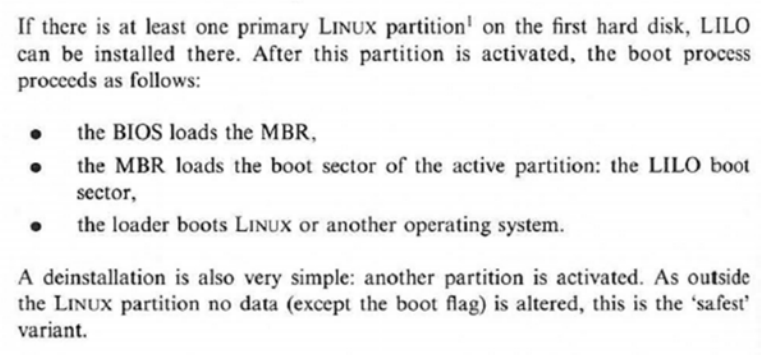
Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

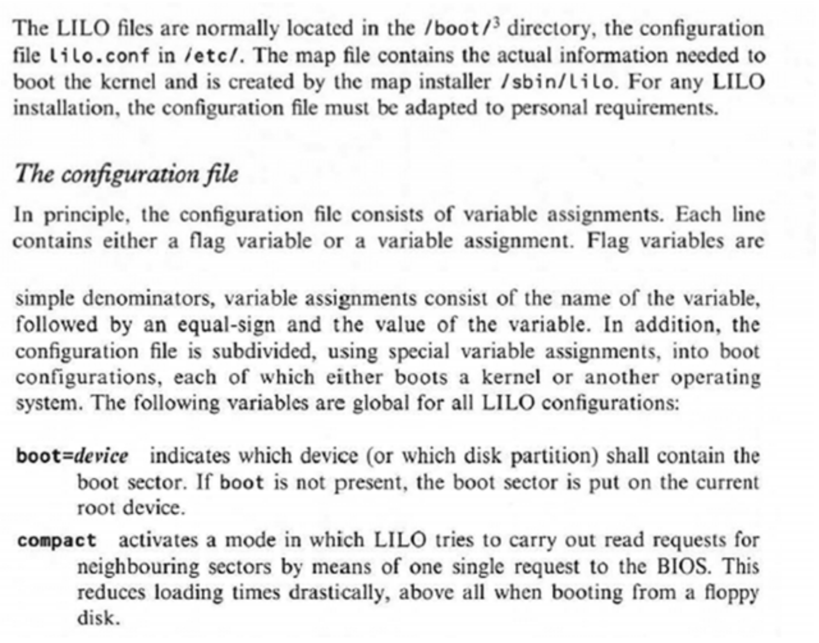
Beck, at Appendix D.1


If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1


The LILO files are normally located in the **/boot/**[3] directory, the configuration file **lilo.conf** in **/etc/**. The map file contains the actual information needed to boot the kernel and is created by the map installer **/sbin/lilo**. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

boot=*device* indicates which device (or which disk partition) shall contain the boot sector. If **boot** is not present, the boot sector is put on the current root device.

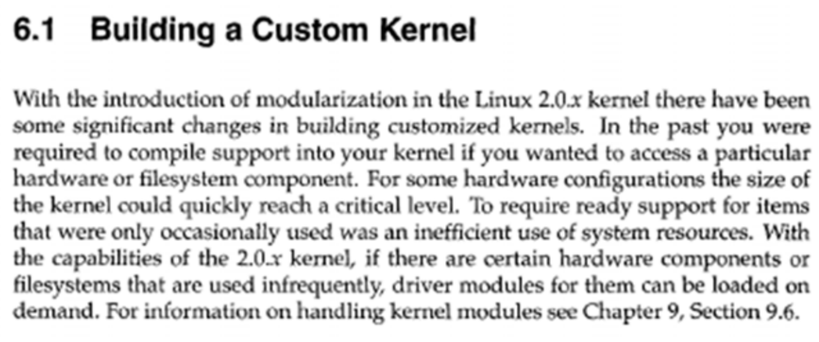
compact activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

1.2 accessing the loaded portion of the boot data in the compressed form from memory;	Linux Kernel, as evidenced by the example citations below, discloses "accessing the loaded portion of the boot data in the compressed form from memory;"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the boot data in the compressed form from memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

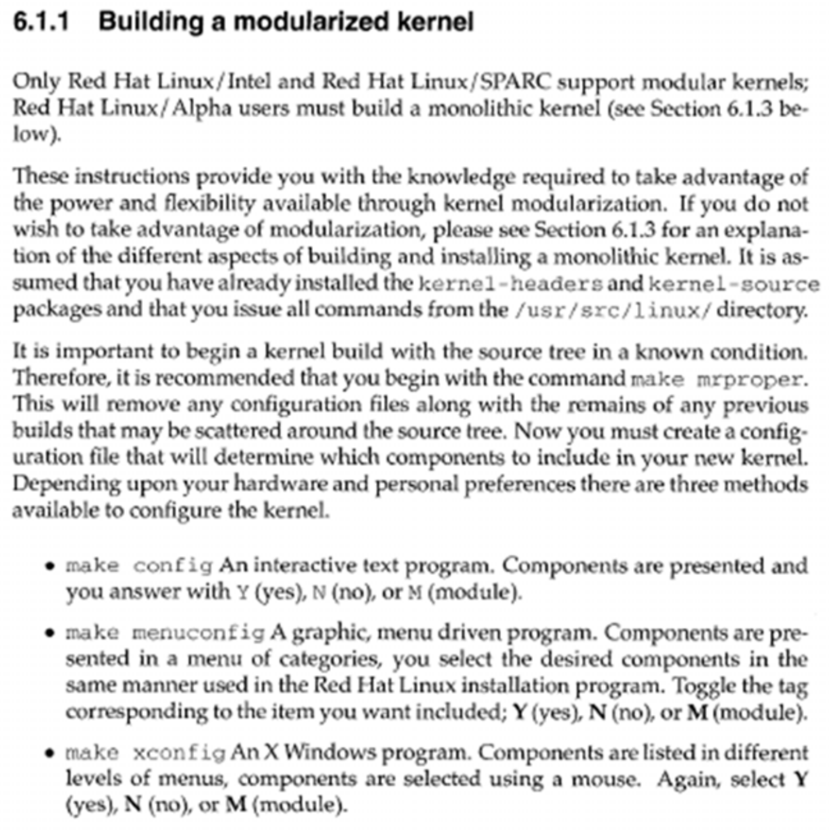
Linux Kernel discloses this limitation:



6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.x kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.x kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

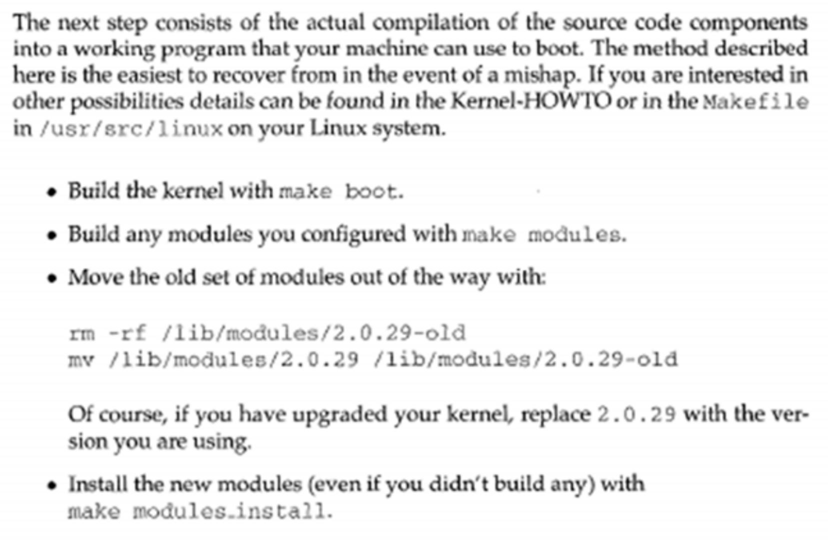
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1



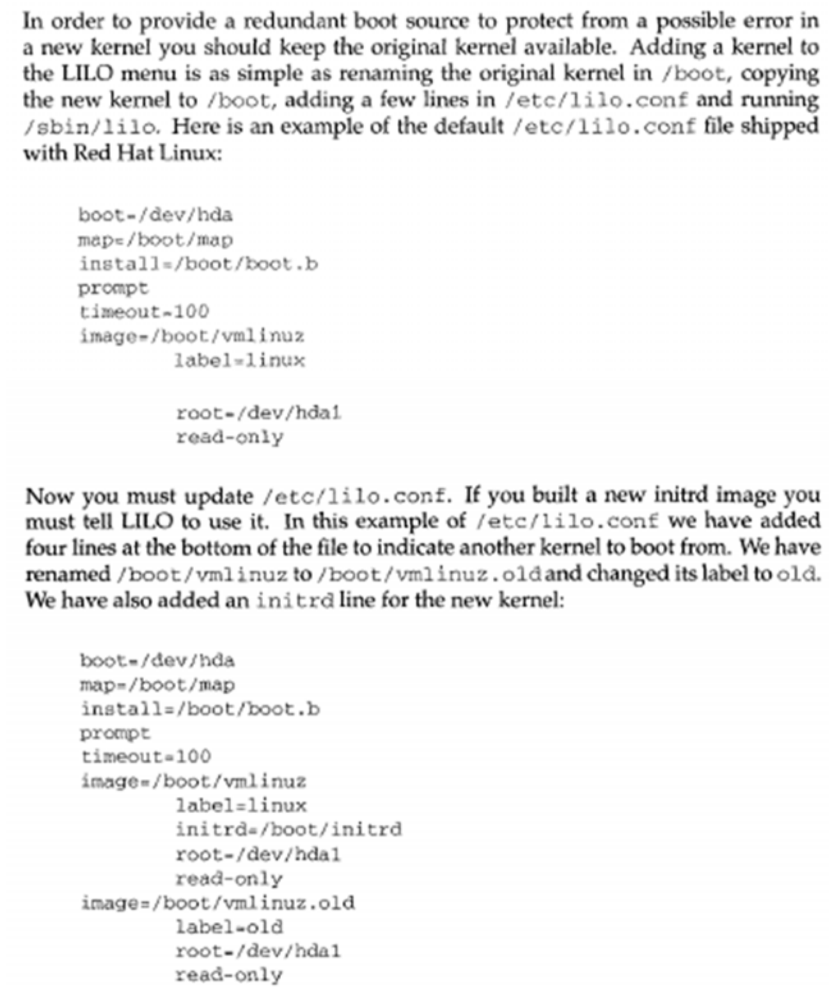
The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1



In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

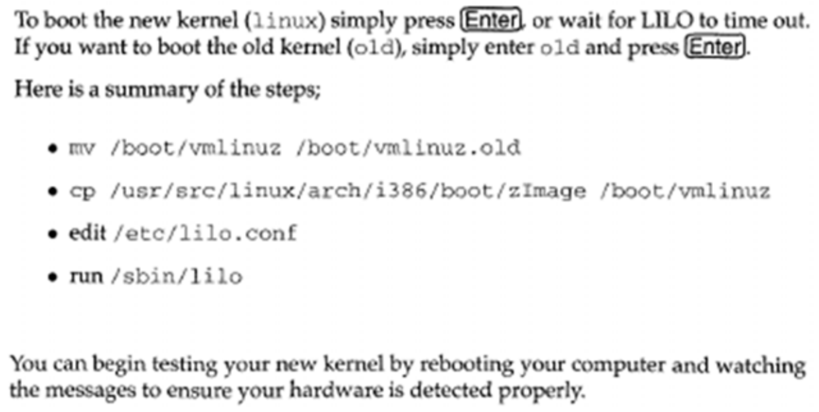
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1



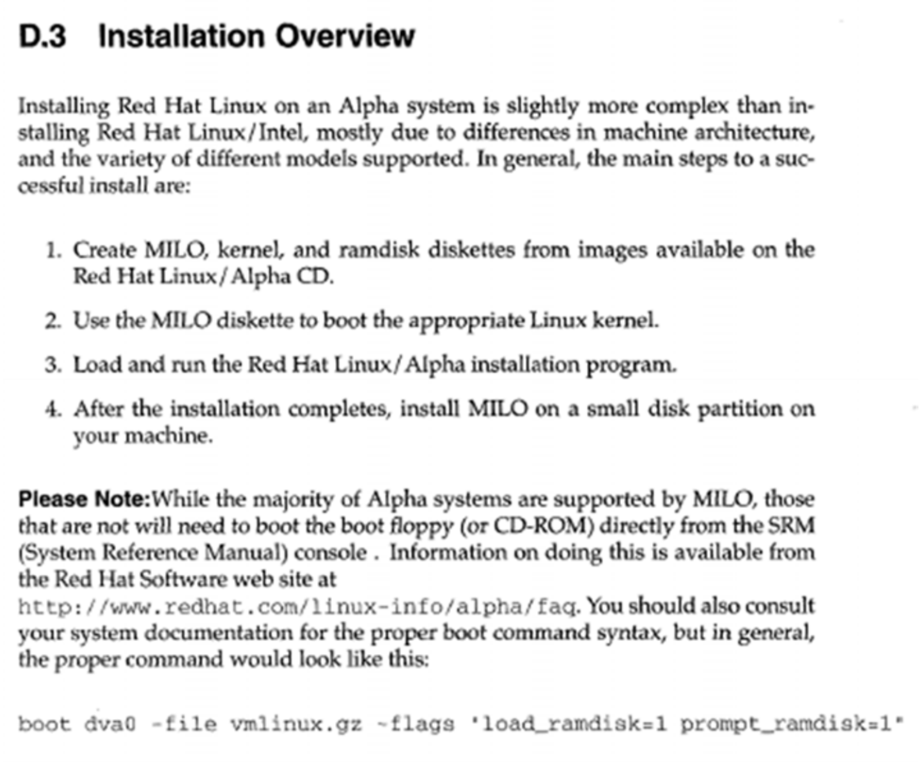
To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1



D.3 Installation Overview

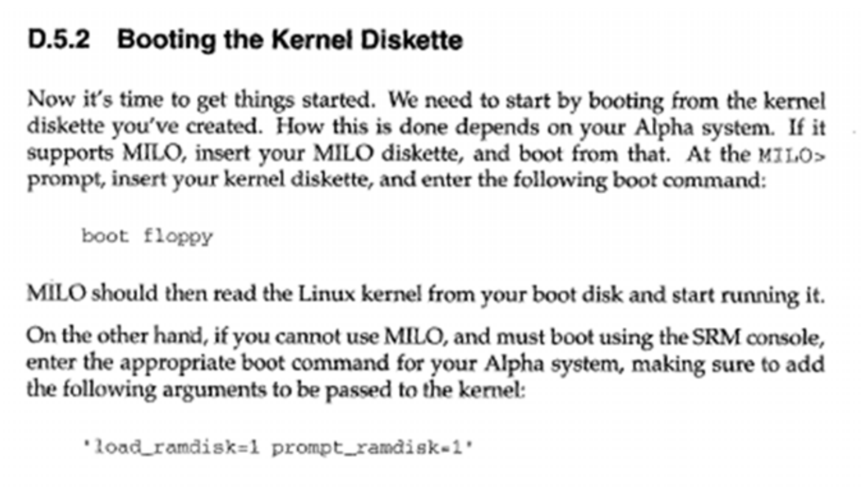
Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3



D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

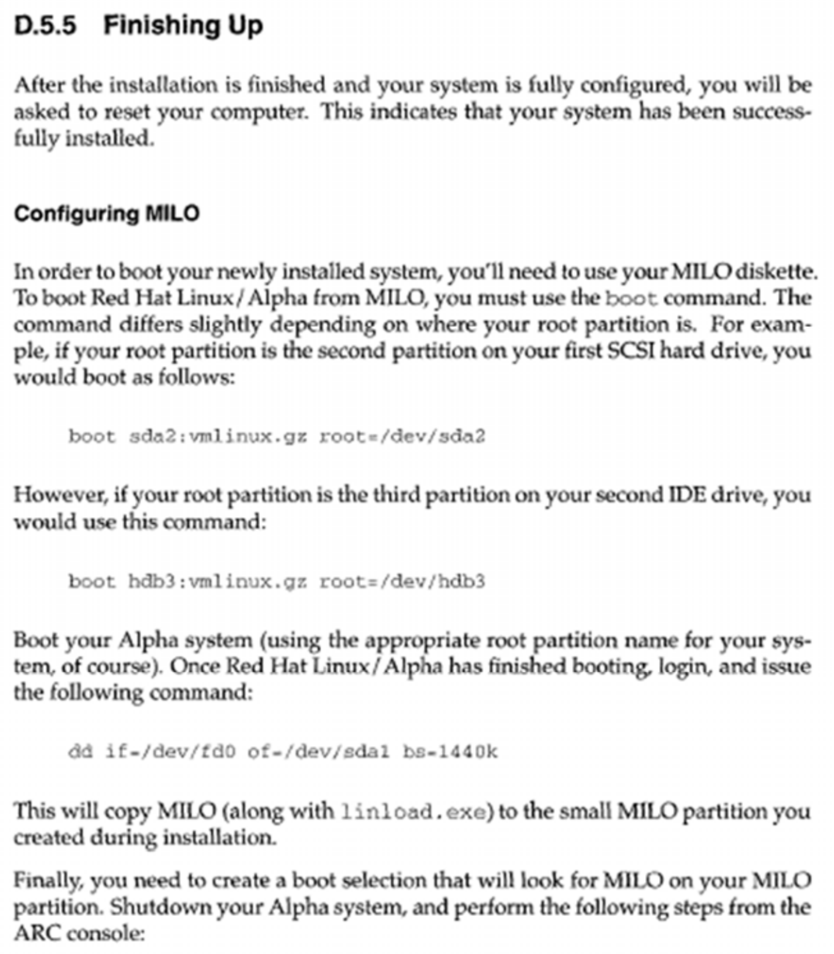
```
     boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
     'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2



D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

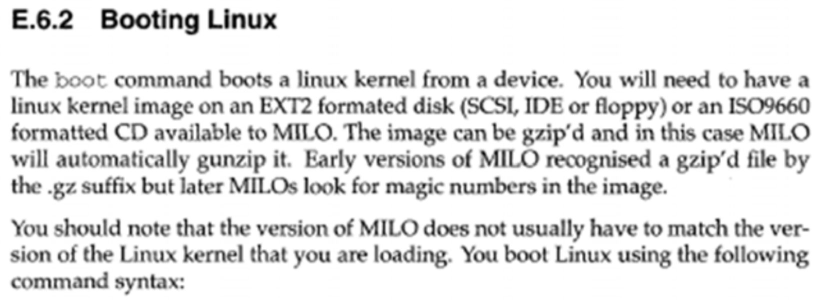
Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

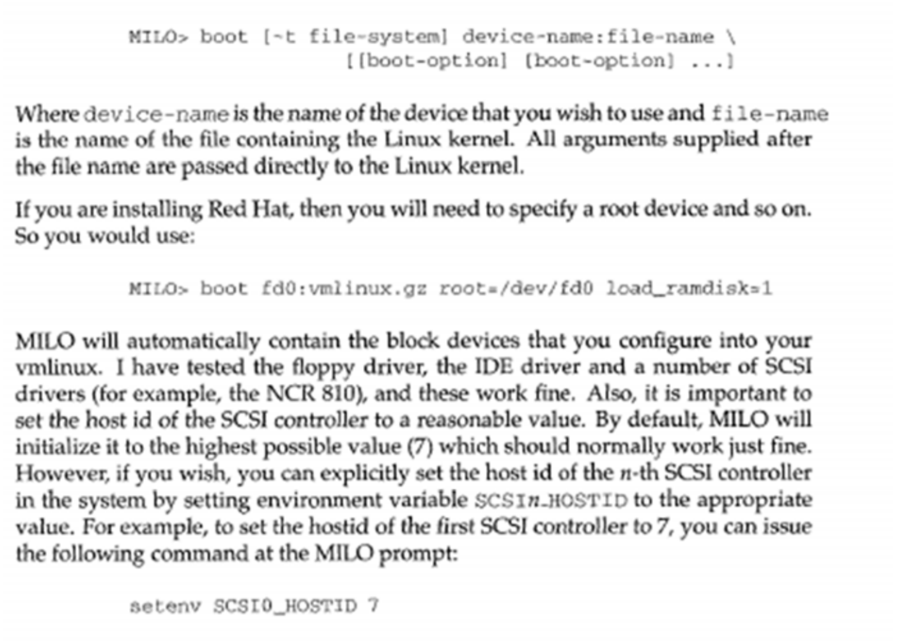
Linux Redhat, at Appendix D5.5



E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:



```
MILO> boot [-t file-system] device-name:file-name \
                  [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

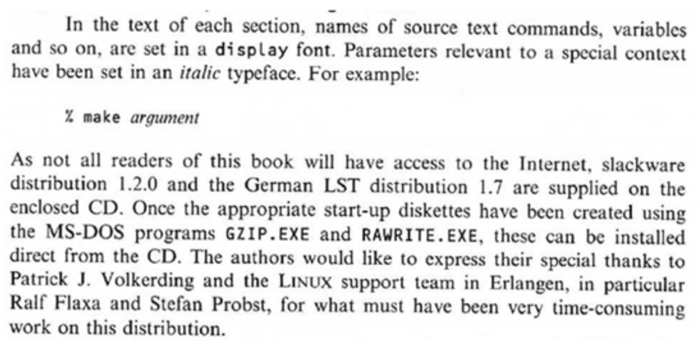
If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

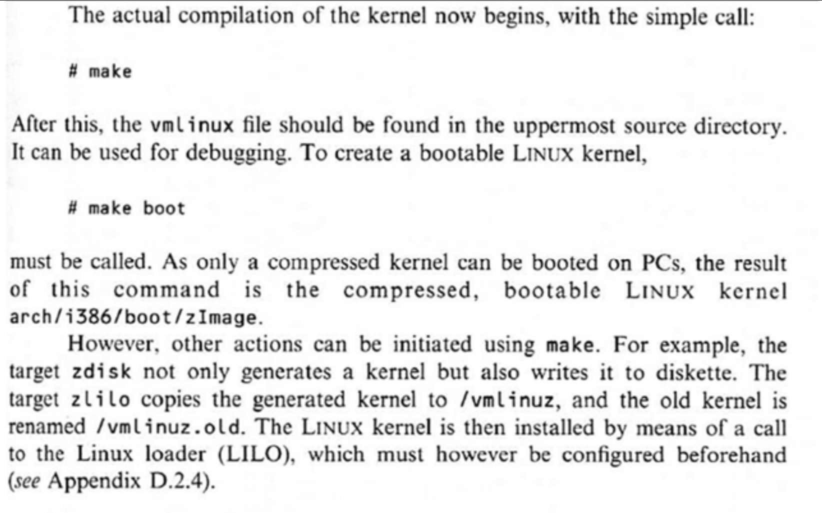


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

`% make` *argument*

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface


The actual compilation of the kernel now begins, with the simple call:

```
# make
```

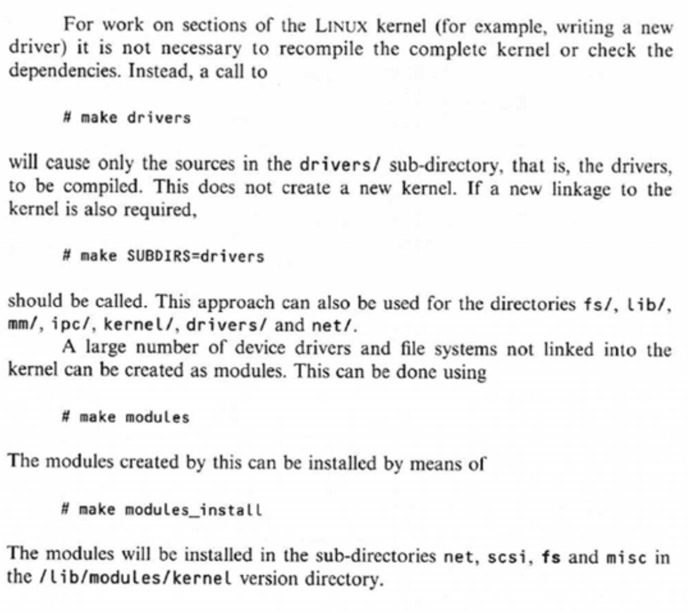
After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2


For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

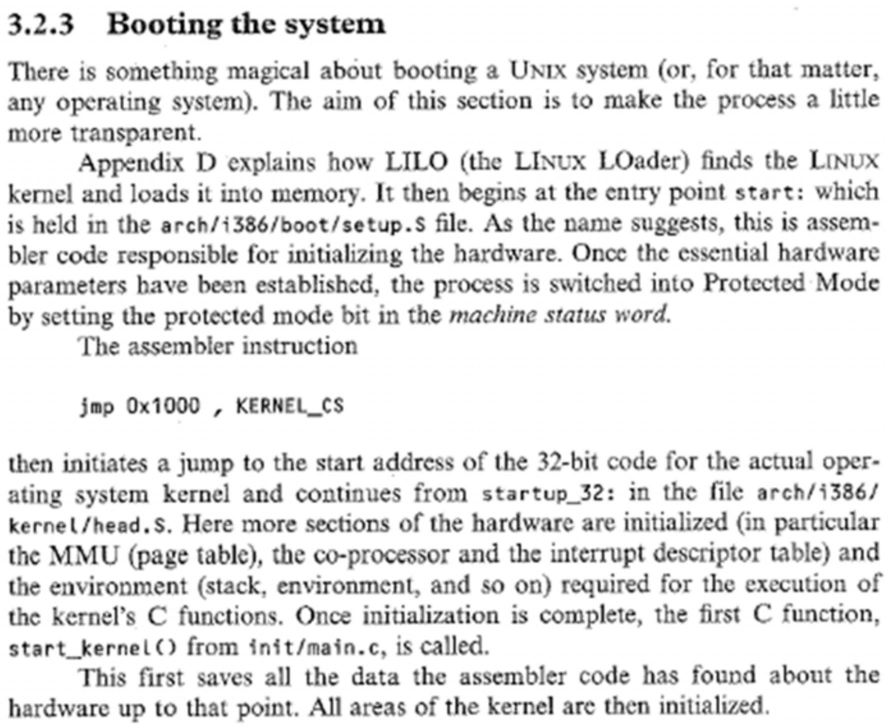
```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2



3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word*.

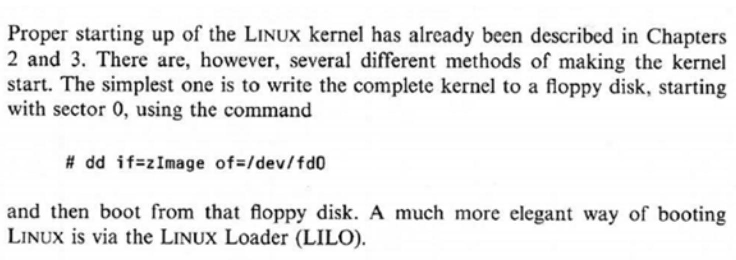
The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

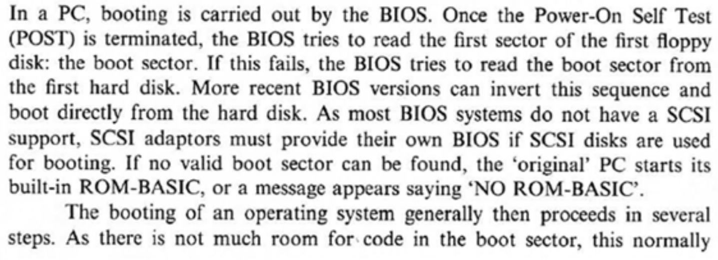


Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D



In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

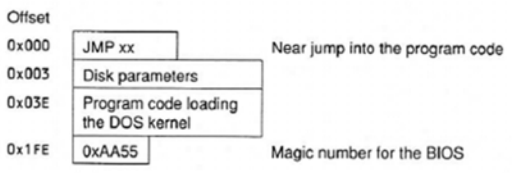

Offset
0x000
JMP xx
Near jump into the program code
0x003
Disk parameters
0x03E
Program code loading the DOS kernel
0x1FE
0xAA55
Magic number for the BIOS

Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

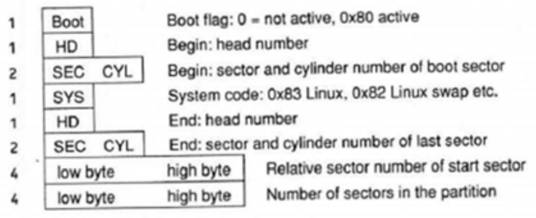

1
Boot
Boot flag: 0 = not active, 0x80 active
1
HD
Begin: head number
2
SEC CYL
Begin: sector and cylinder number of boot sector
1
SYS
System code: 0x83 Linux, 0x82 Linux swap etc.
1
HD
End: head number
2
SEC CYL
End: sector and cylinder number of last sector
4
low byte high byte
Relative sector number of start sector
4
low byte high byte
Number of sectors in the partition

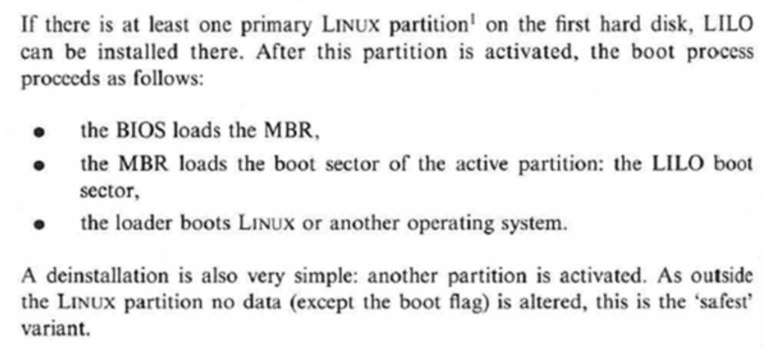
Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

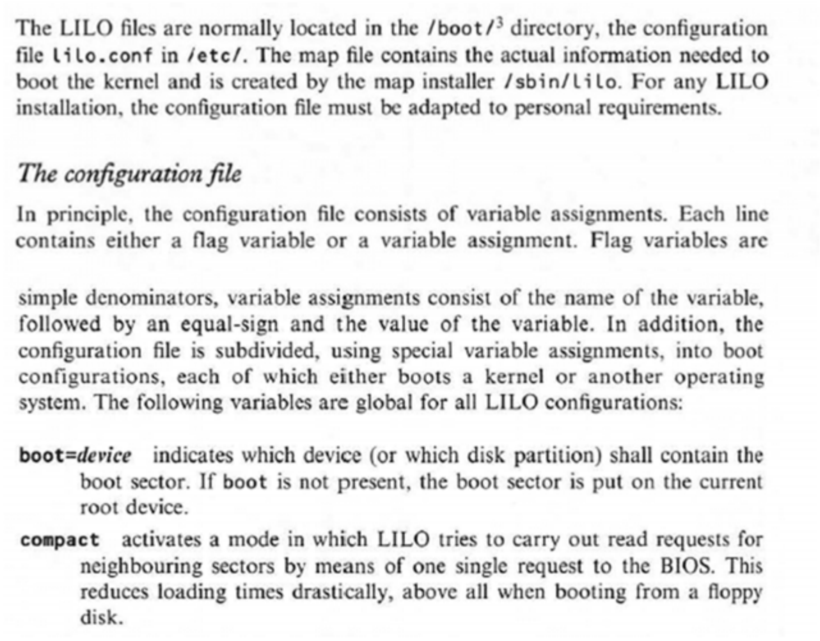


If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1



The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

`boot=`*device* indicates which device (or which disk partition) shall contain the boot sector. If **`boot`** is not present, the boot sector is put on the current root device.

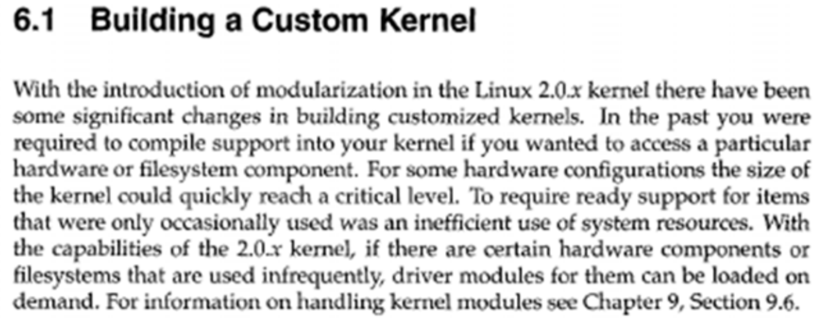
`compact` activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

1.3 decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and	Linux Kernel, as evidenced by the example citations below, discloses "decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form; and"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

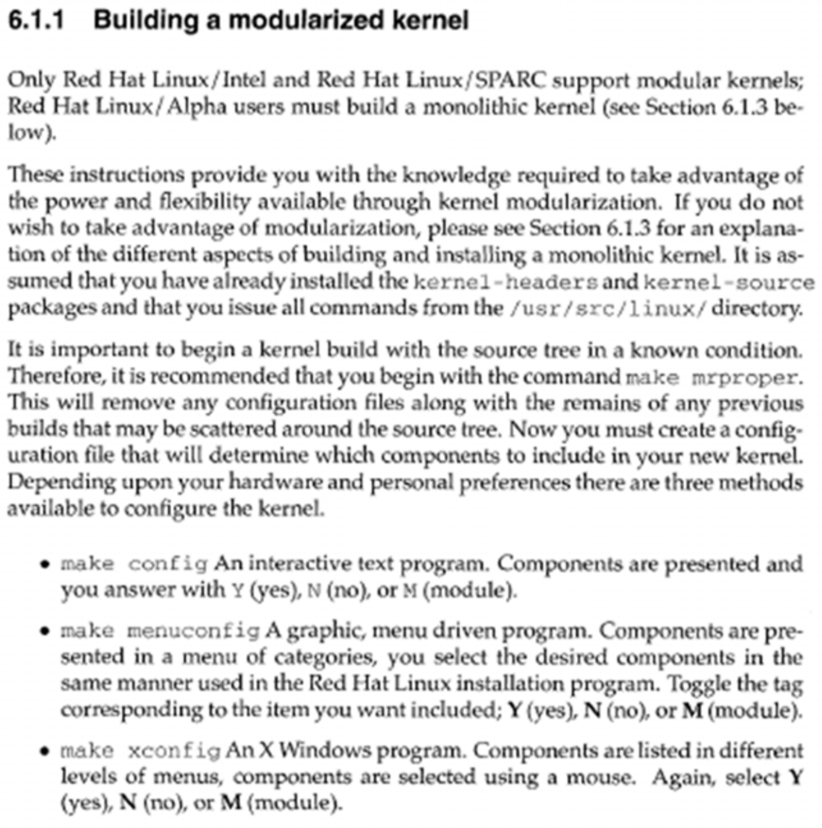
Linux Kernel discloses this limitation:



6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

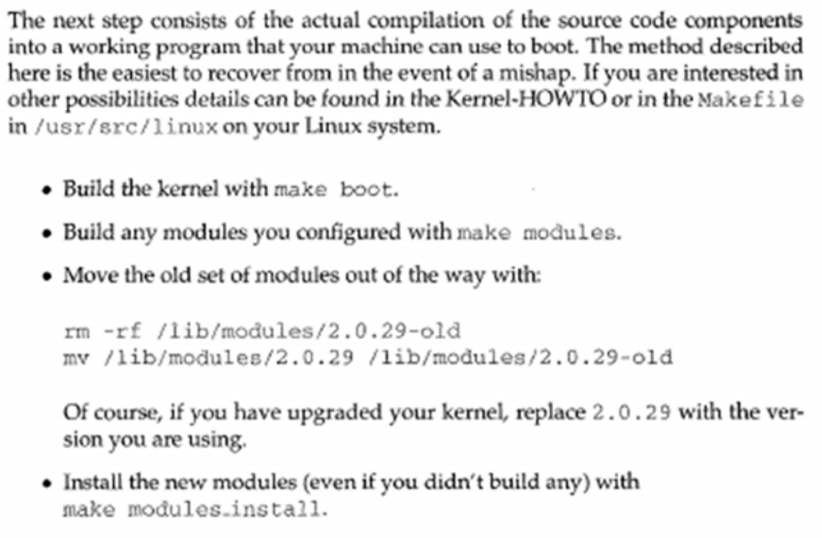
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1



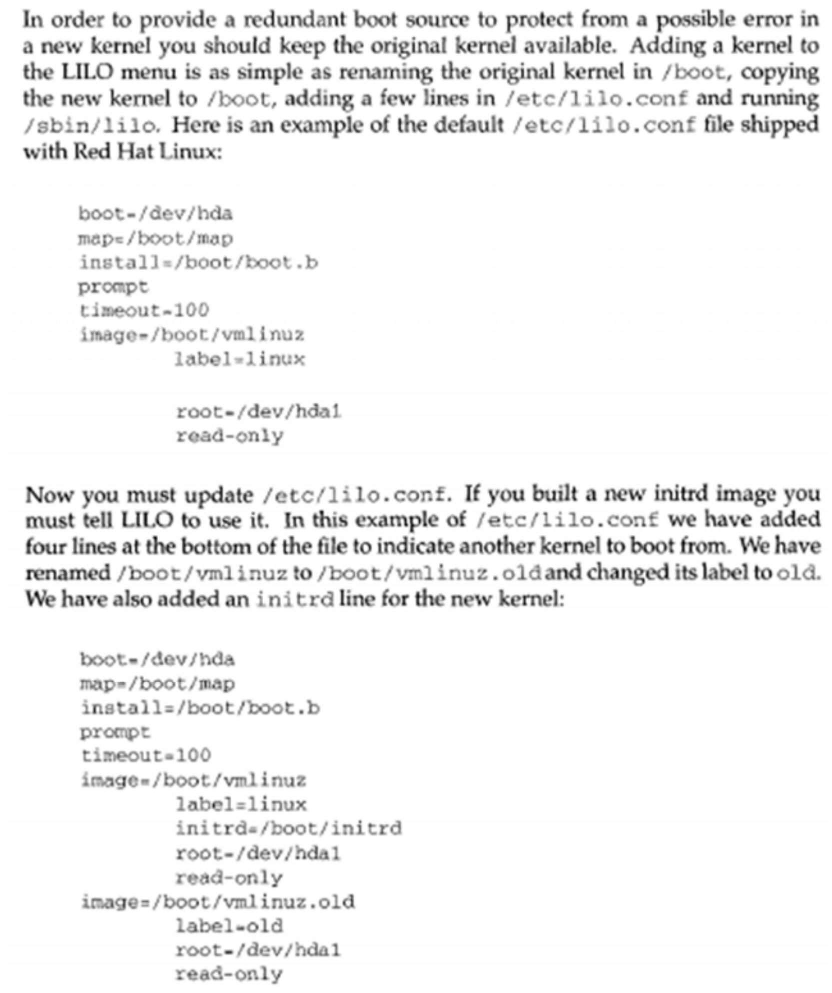
The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1



In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

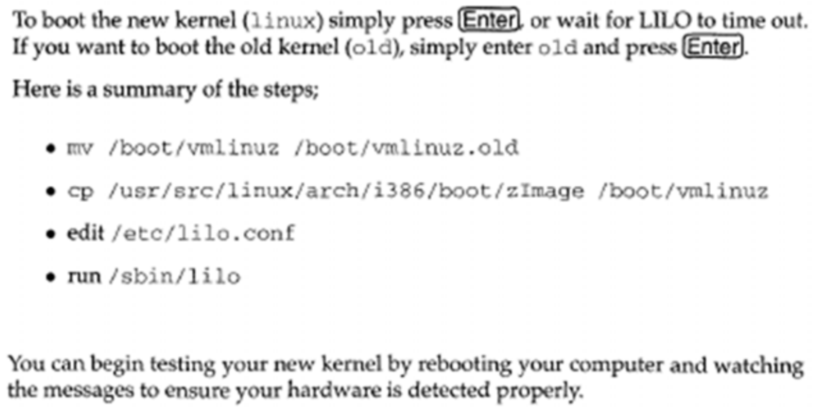
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1



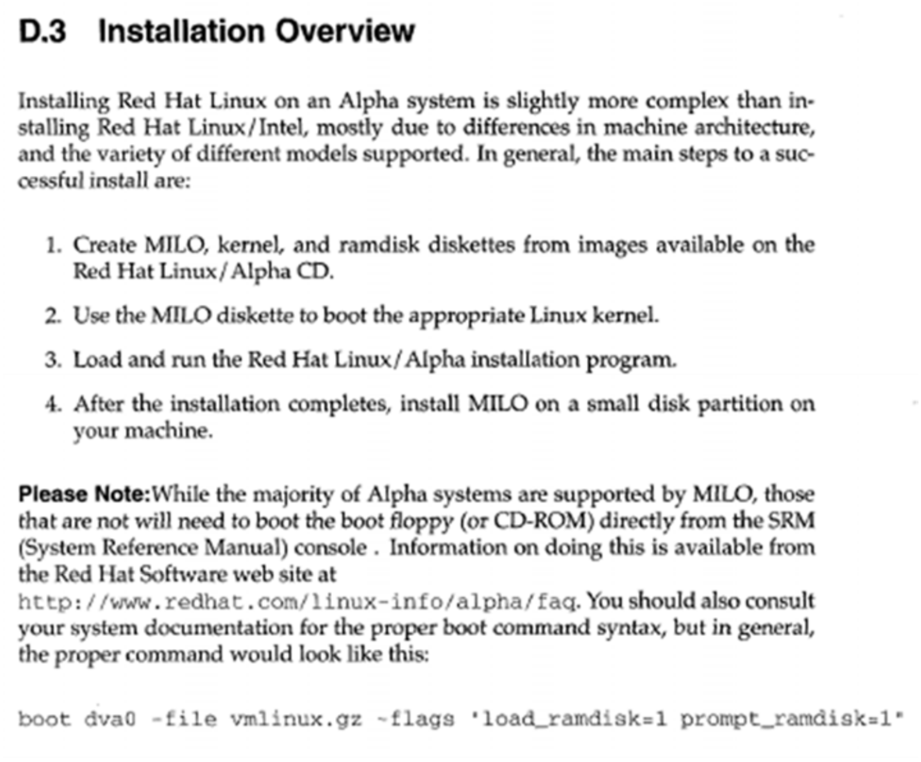
To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1



D.3 Installation Overview

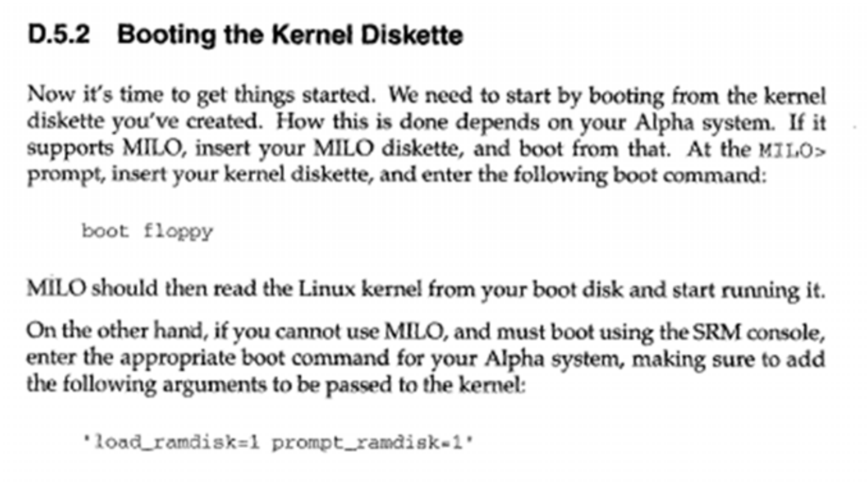
Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3



D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

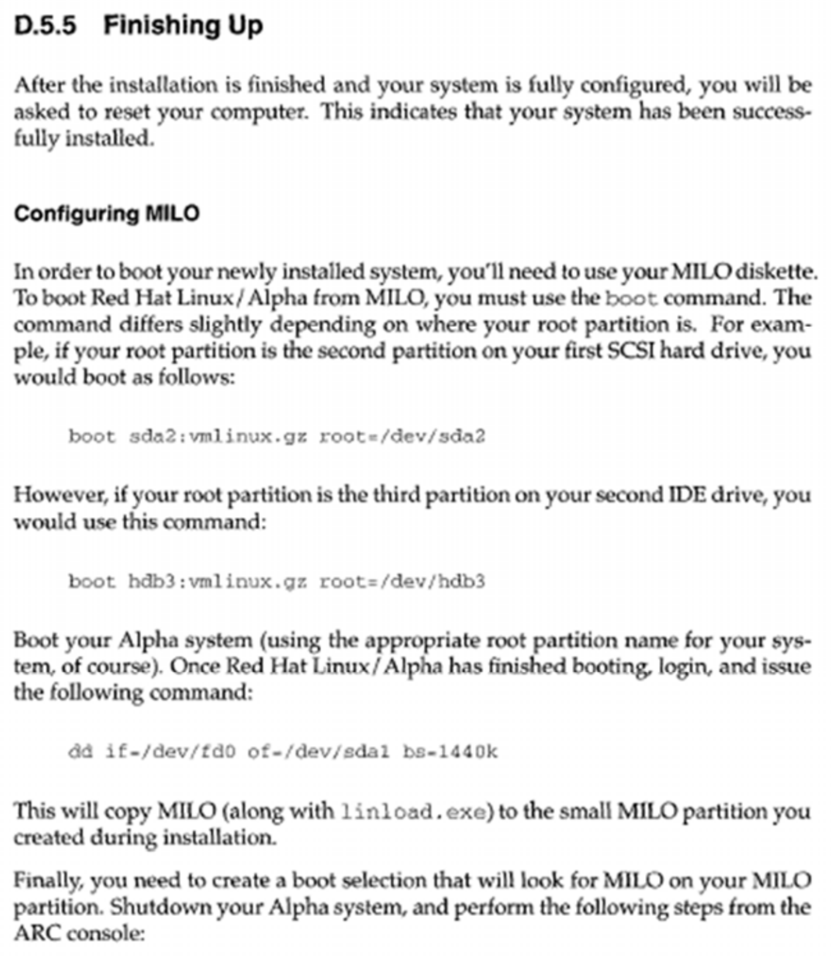
```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2



D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

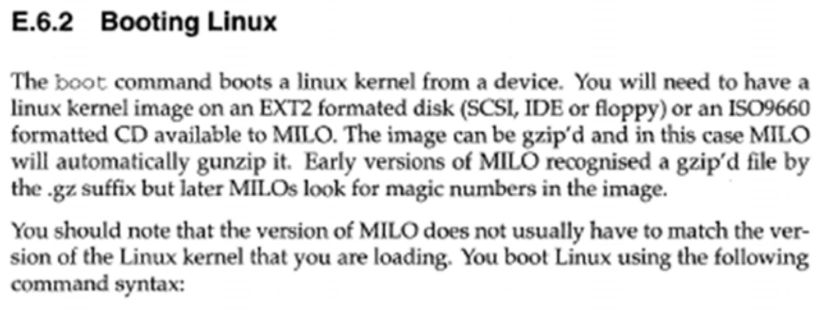
Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

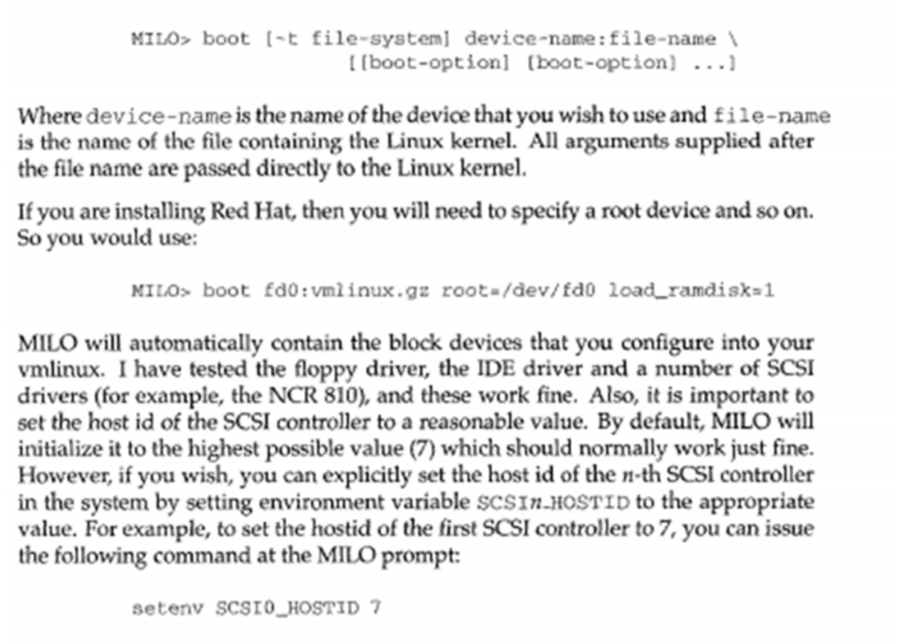
Linux Redhat, at Appendix D5.5



E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:



```
        MILO> boot [-t file-system] device-name:file-name \
                      [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

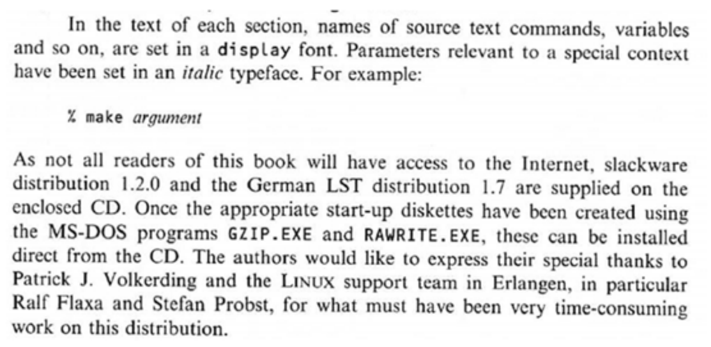
If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
        MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
        setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

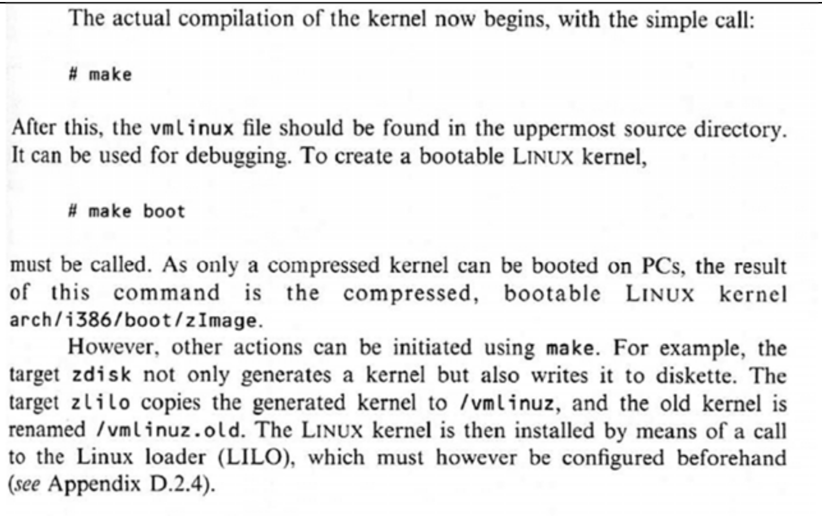


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface



The actual compilation of the kernel now begins, with the simple call:

```
# make
```

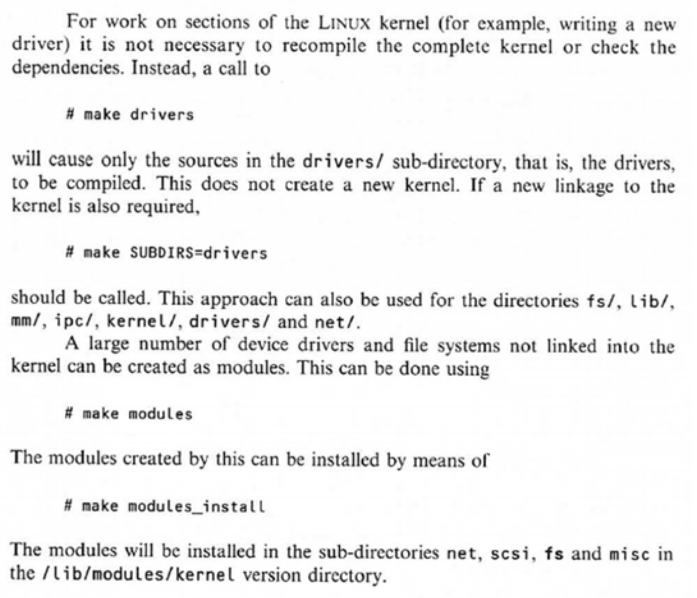
After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using **make**. For example, the target **zdisk** not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2



For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

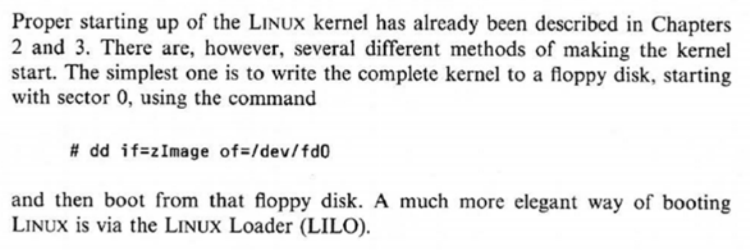
```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

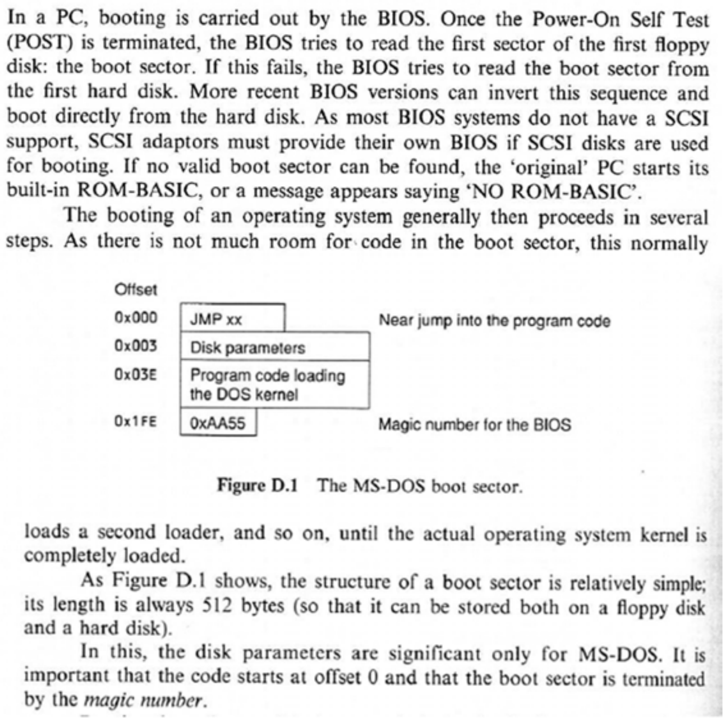


Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D



In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally loads a second loader, and so on, until the actual operating system kernel is completely loaded.

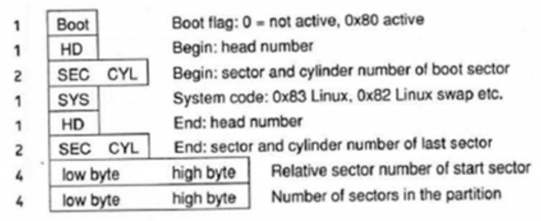
Offset
0x000
JMP xx
Near jump into the program code
0x003
Disk parameters
0x03E
Program code loading the DOS kernel
0x1FE
0xAA55
Magic number for the BIOS

Figure D.1 The MS-DOS boot sector.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

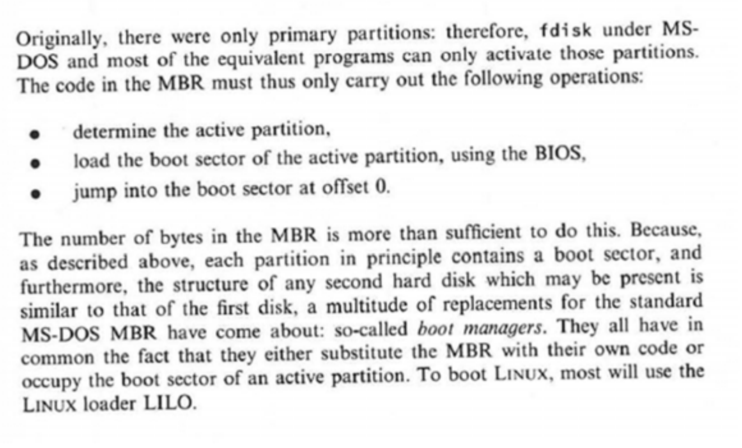
In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1



1	Boot	Boot flag: 0 = not active, 0x80 active
1	HD	Begin: head number
2	SEC CYL	Begin: sector and cylinder number of boot sector
1	SYS	System code: 0x83 Linux, 0x82 Linux swap etc.
1	HD	End: head number
2	SEC CYL	End: sector and cylinder number of last sector
4	low byte high byte	Relative sector number of start sector
4	low byte high byte	Number of sectors in the partition

Figure D.3 Structure of a partition entry.

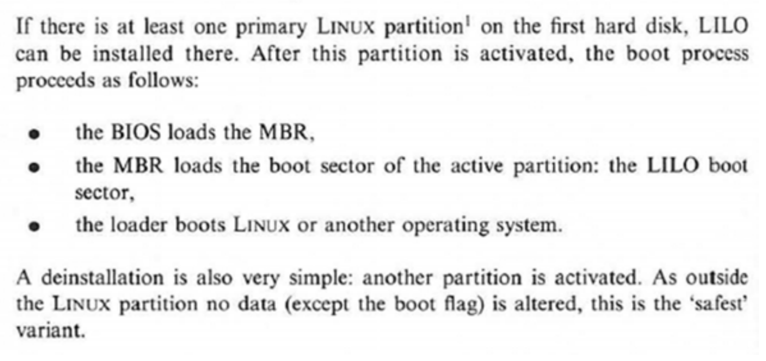


Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

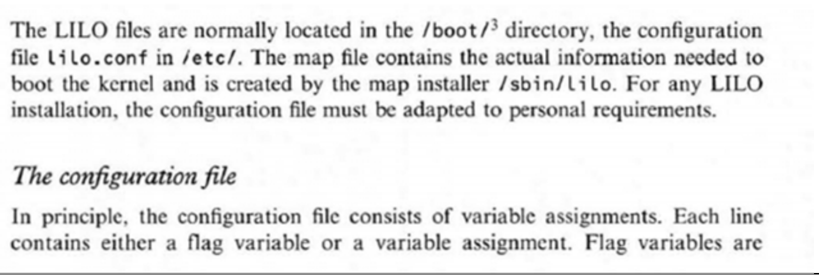


If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

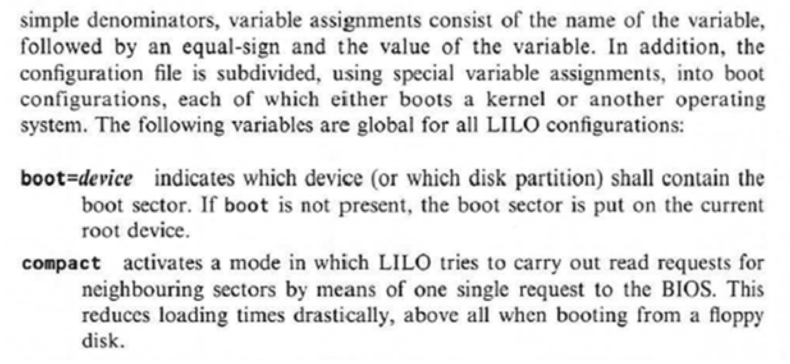
Beck, at Appendix D.2.1



The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are


simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

boot=*device* indicates which device (or which disk partition) shall contain the boot sector. If **boot** is not present, the boot sector is put on the current root device.

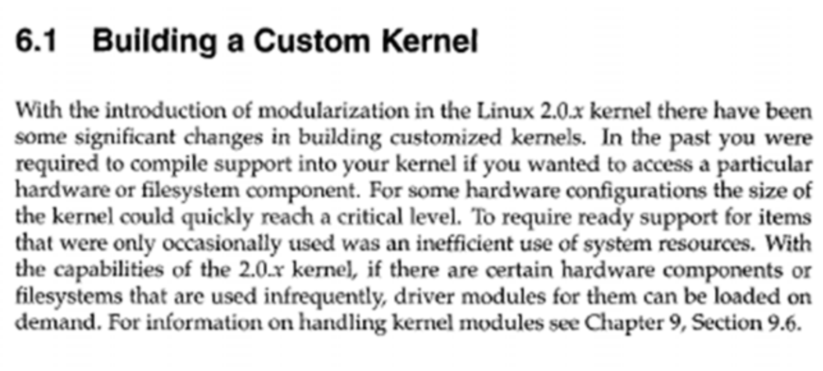
compact activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

1.4.1 updating the boot data list,	Linux Kernel, as evidenced by the example citations below, discloses "updating the boot data list,"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

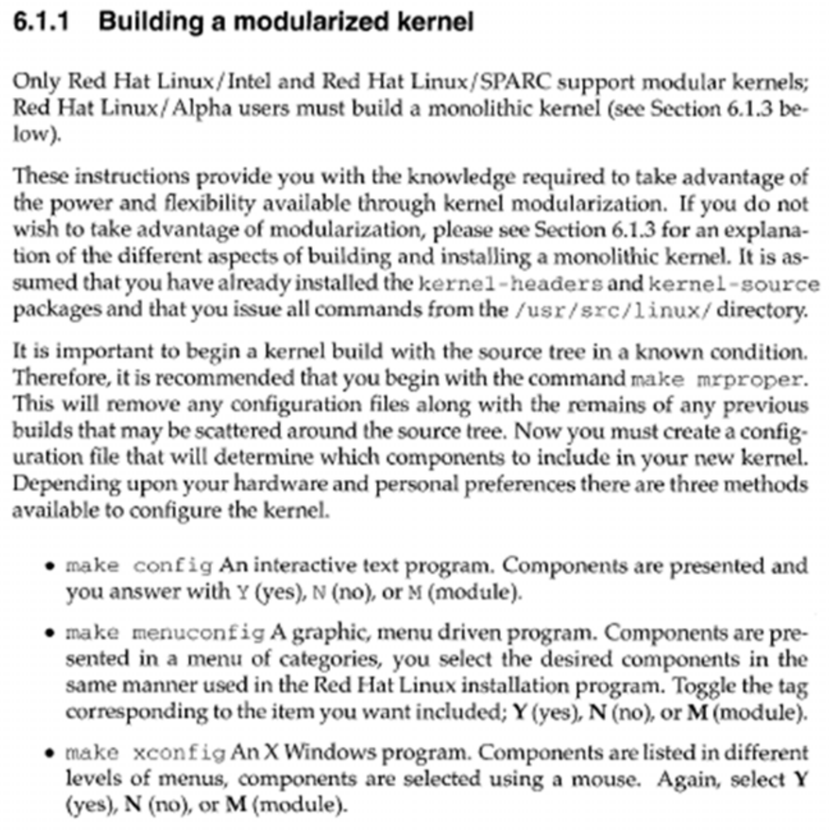
Linux Kernel discloses this limitation:



6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.x kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.x kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

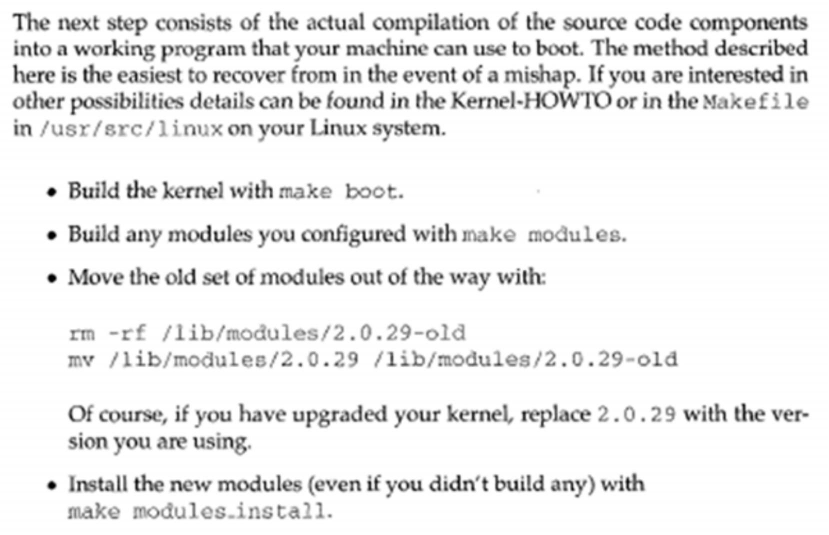
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1



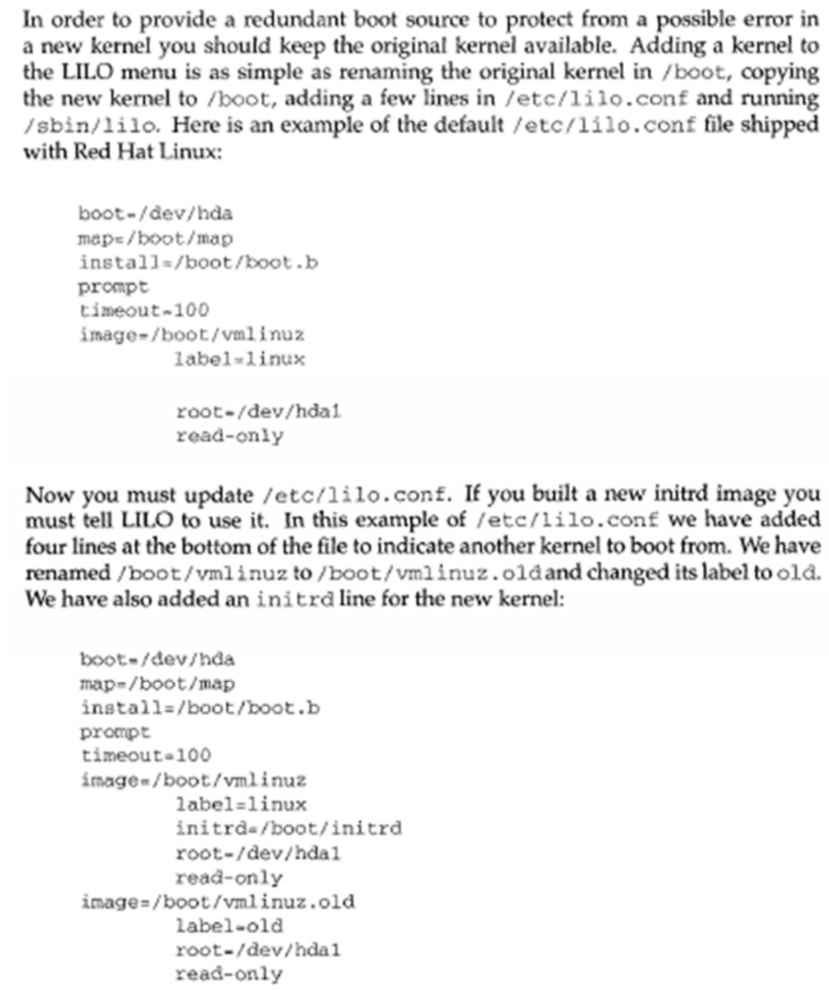
The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1



In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

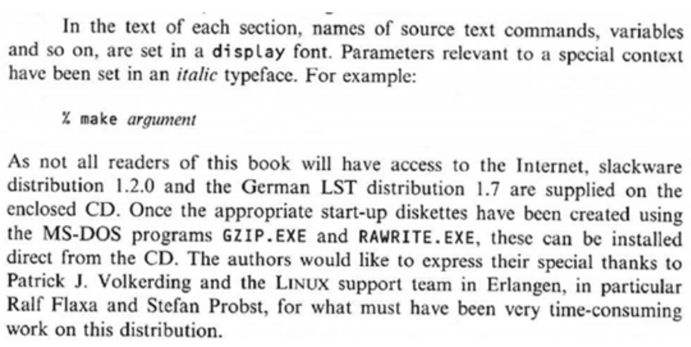
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1

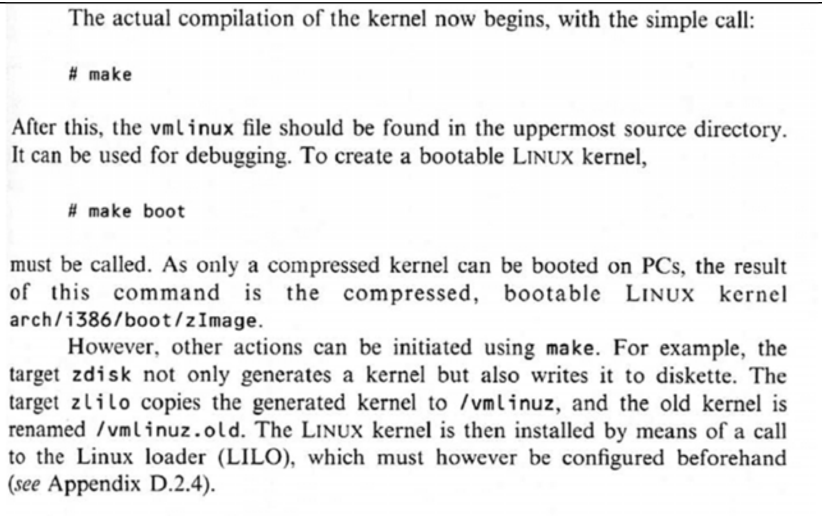


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface



The actual compilation of the kernel now begins, with the simple call:

```
# make
```

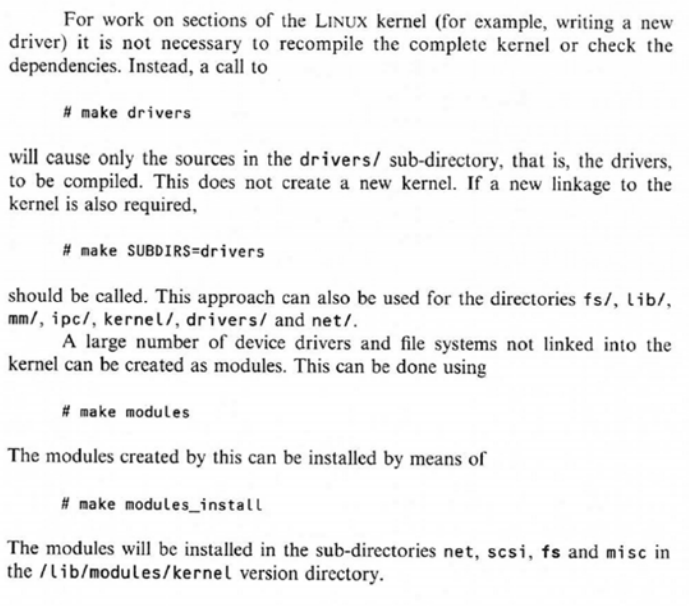
After this, the vmlinux file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel arch/i386/boot/zImage.

However, other actions can be initiated using make. For example, the target zdisk not only generates a kernel but also writes it to diskette. The target zlilo copies the generated kernel to /vmlinuz, and the old kernel is renamed /vmlinuz.old. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2



For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the drivers/ sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories fs/, lib/, mm/, ipc/, kernel/, drivers/ and net/.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

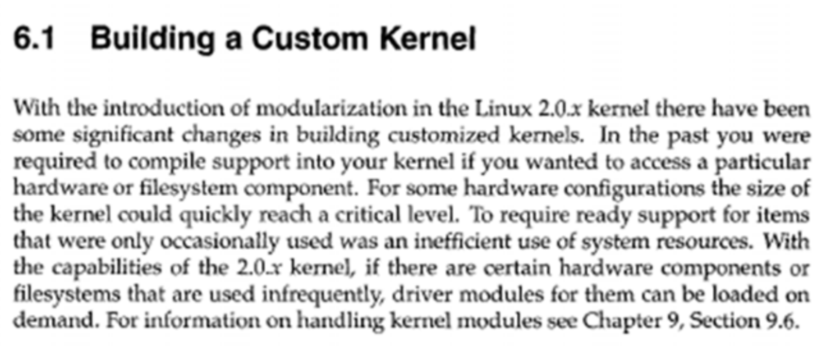
The modules will be installed in the sub-directories net, scsi, fs and misc in the /lib/modules/kernel version directory.

Beck, at 2.2

1.4.2 wherein the decompressed portion of boot data comprises a portion of the operating system.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the decompressed portion of boot data comprises a portion of the operating system."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the decompressed portion of boot data comprises a portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

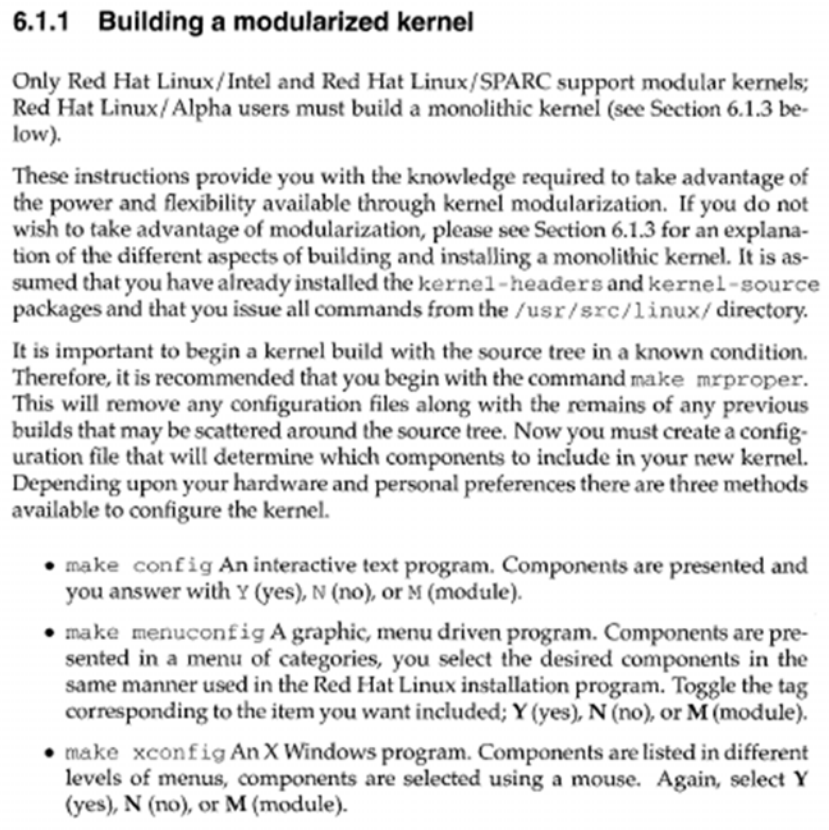
Linux Kernel discloses this limitation:



6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

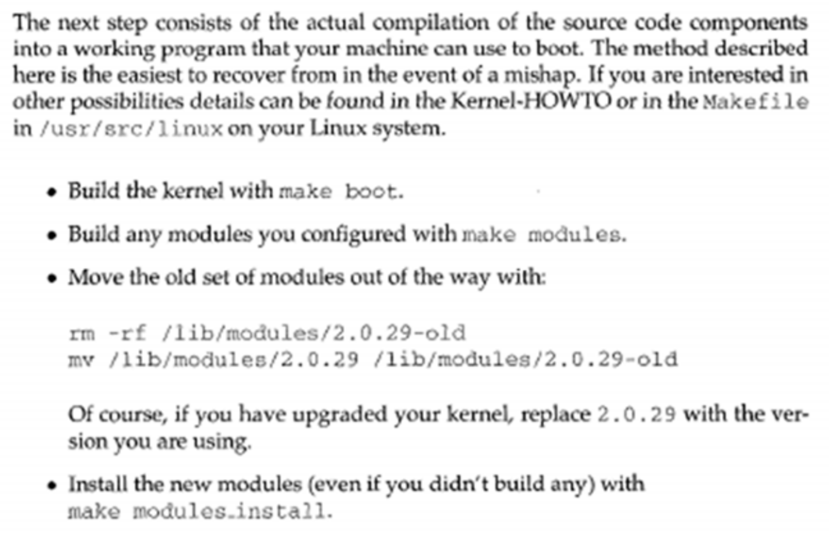
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1



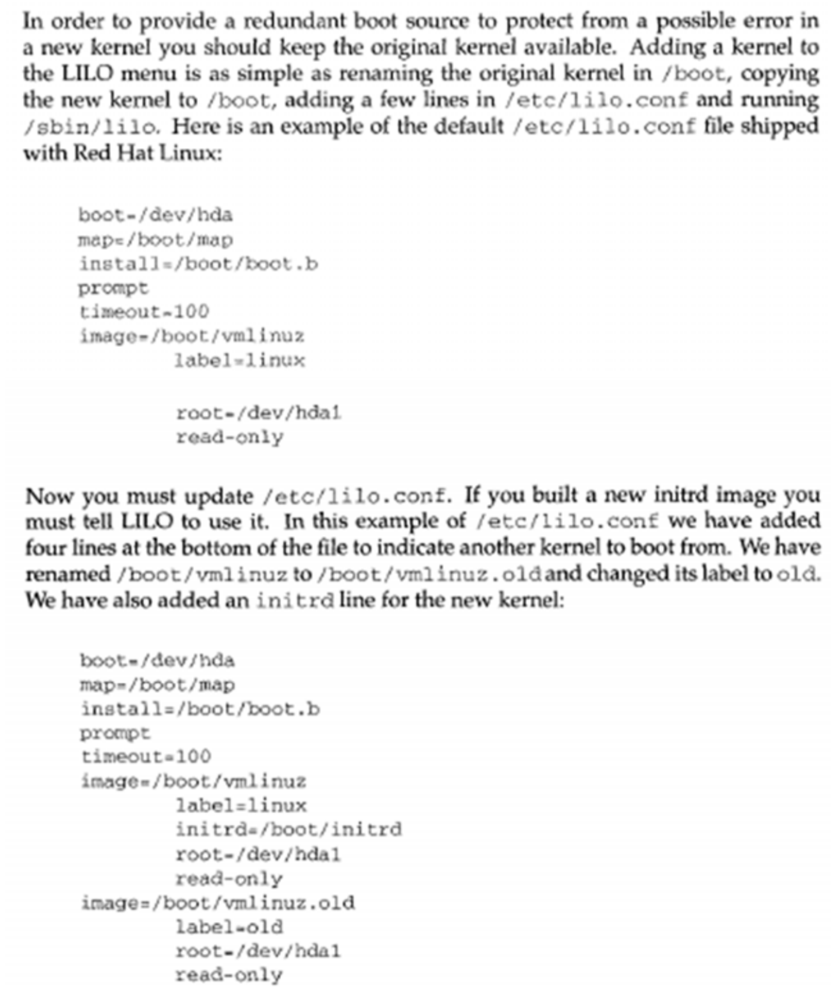
The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1



In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

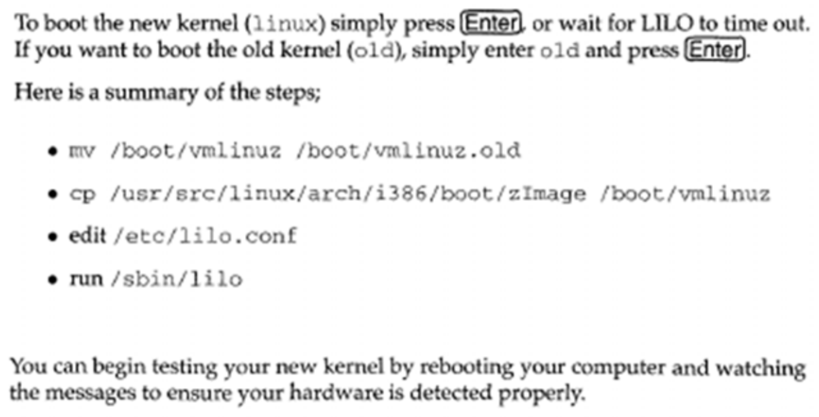
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1



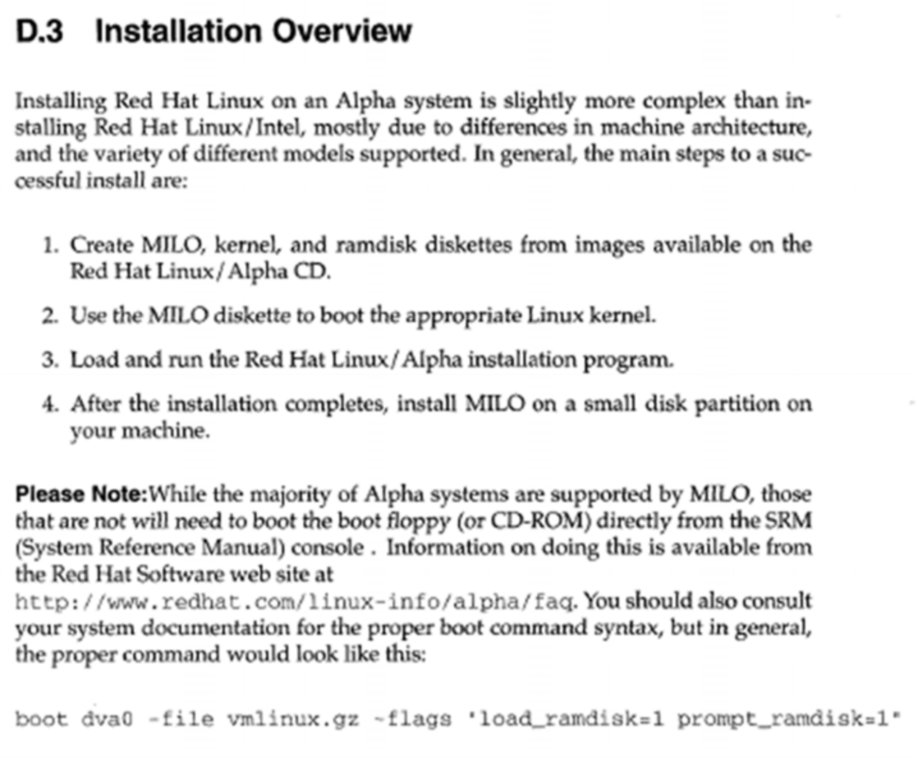
To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1



D.3 Installation Overview

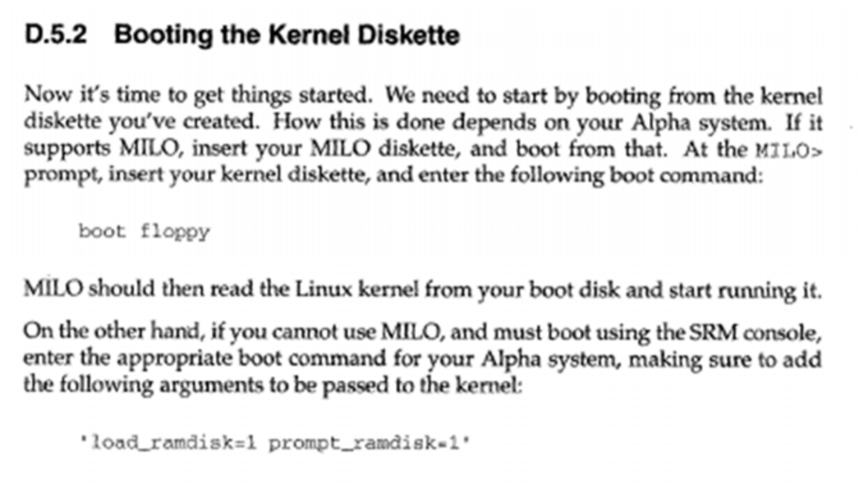
Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3



D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

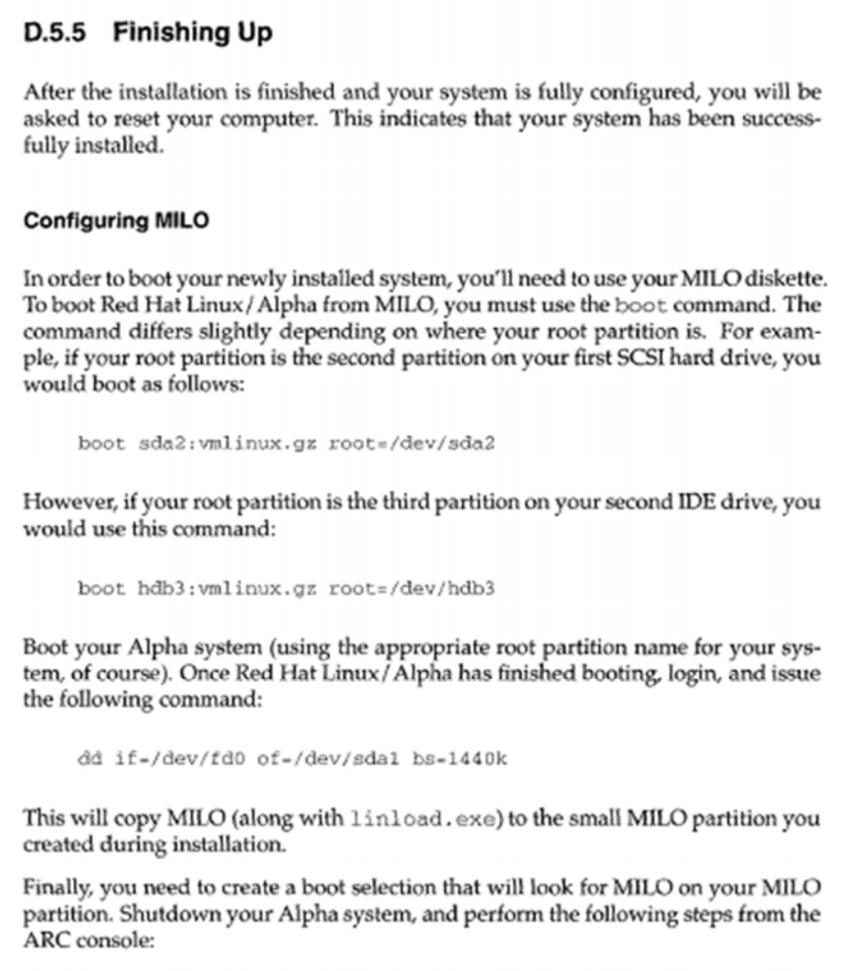
```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2



D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

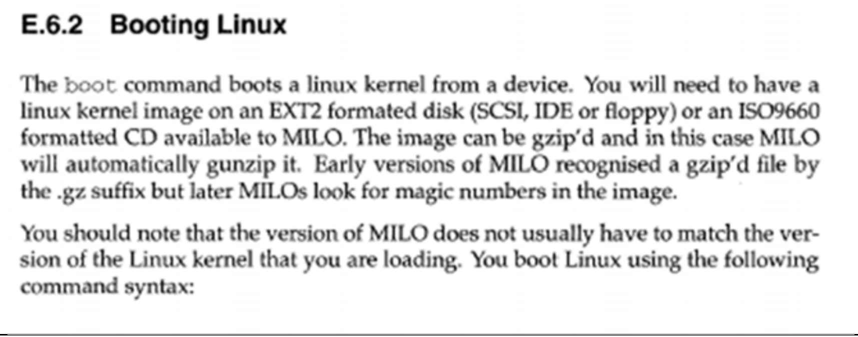
Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

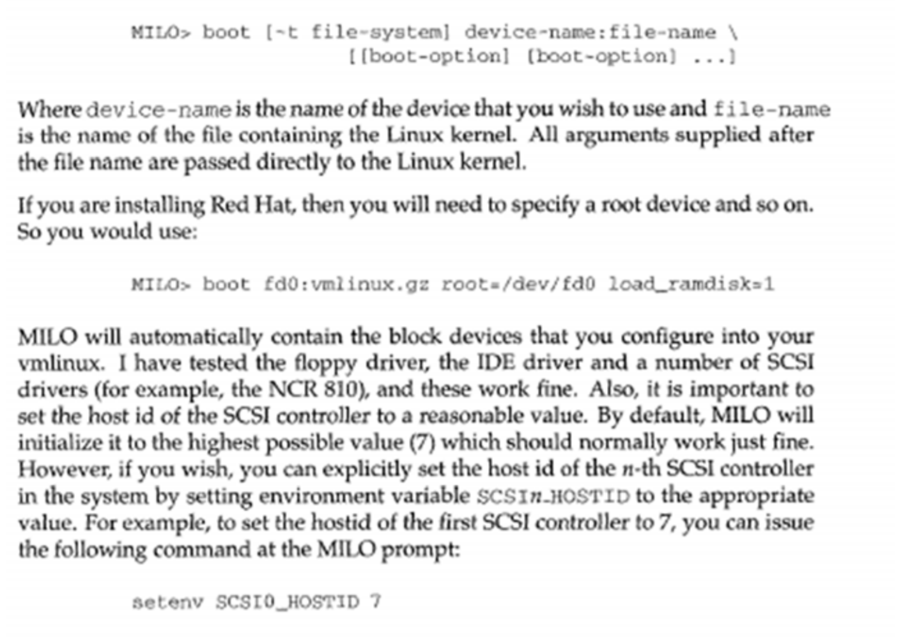
Linux Redhat, at Appendix D5.5



E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:



```
MILO> boot [-t file-system] device-name:file-name \
                 [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

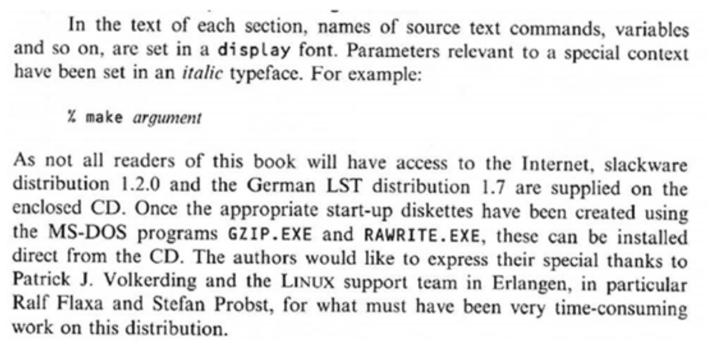
If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

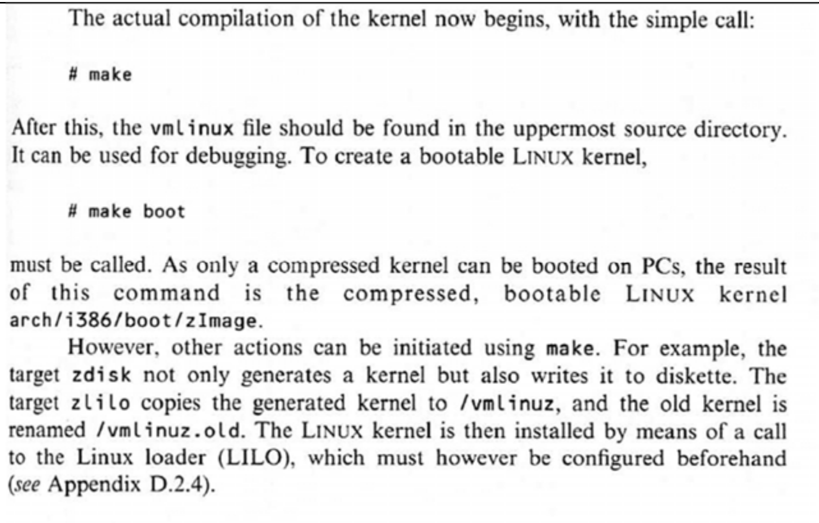


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

`% make` *argument*

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface



The actual compilation of the kernel now begins, with the simple call:

```
# make
```

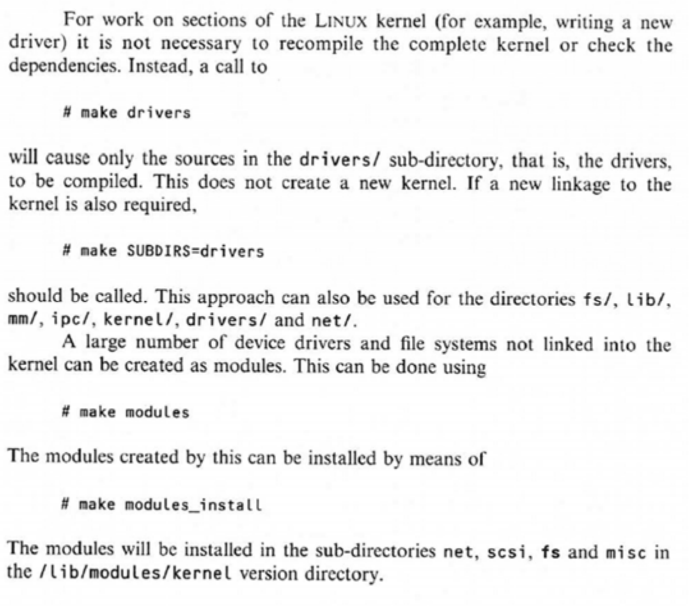
After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2



For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

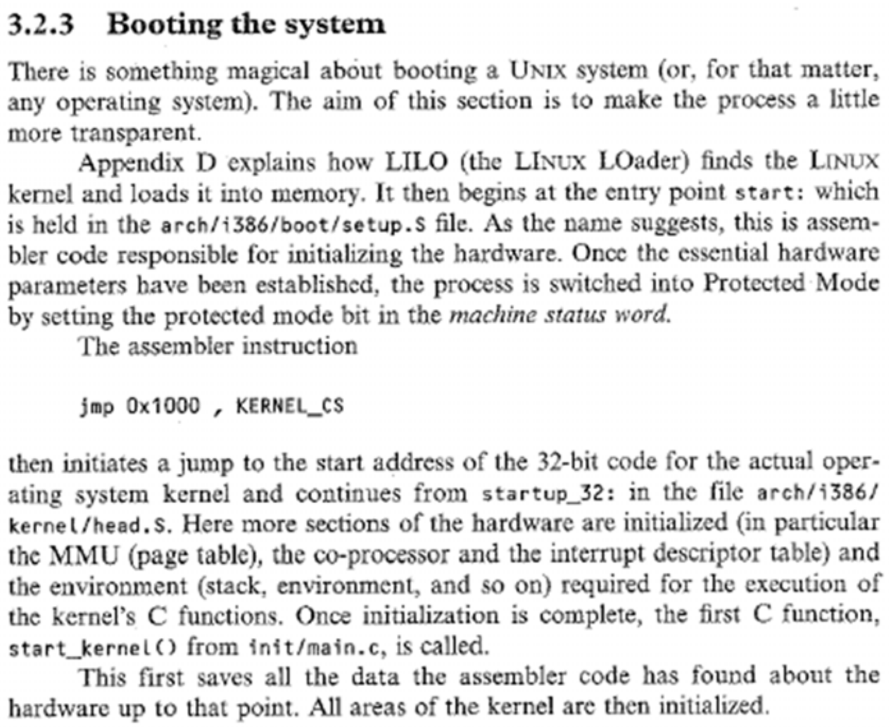
```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2



3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word.*

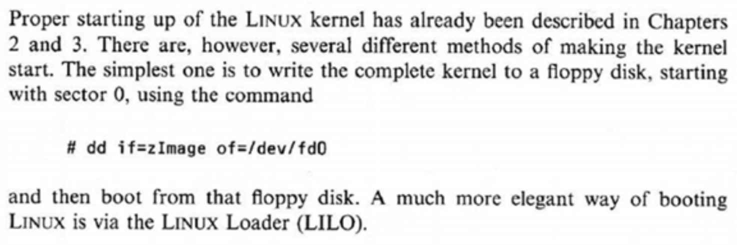
The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

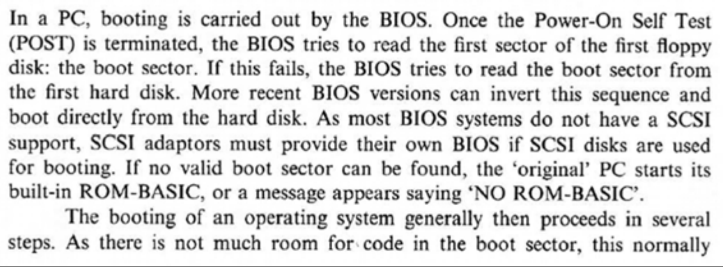


Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

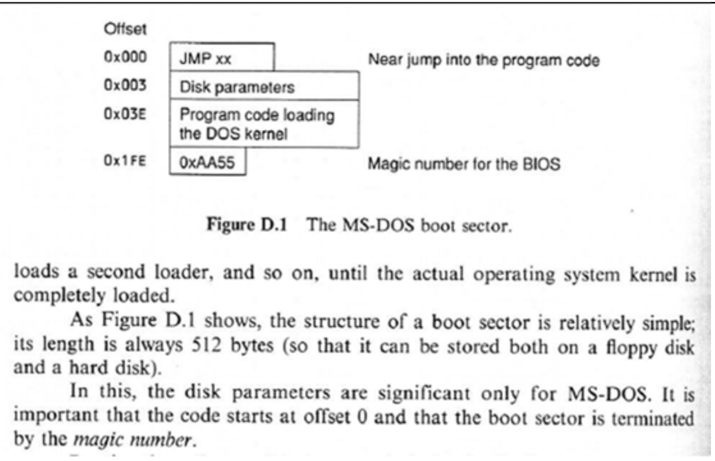
and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D



In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally



Offset		
0x000	JMP xx	Near jump into the program code
0x003	Disk parameters	
0x03E	Program code loading the DOS kernel	
0x1FE	0xAA55	Magic number for the BIOS

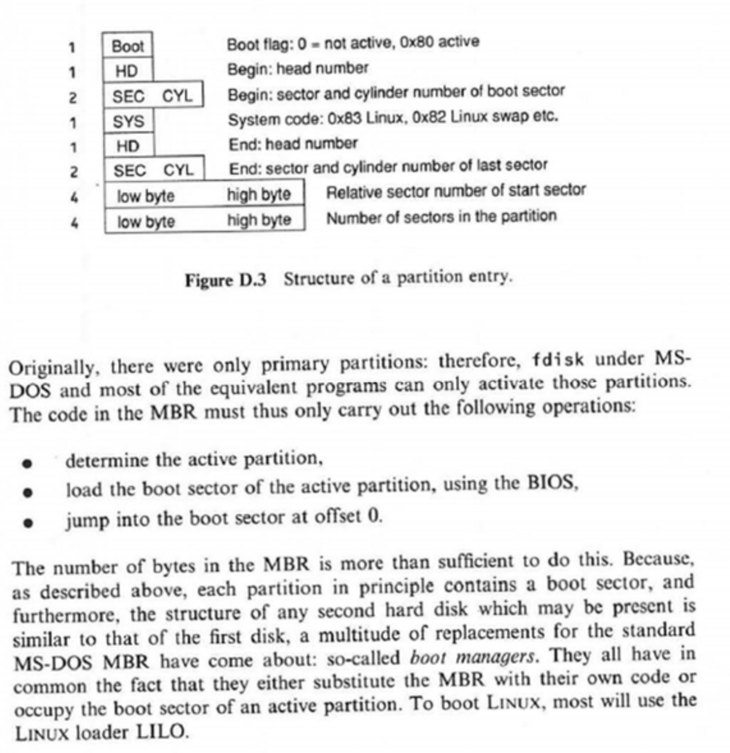
Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1



1	Boot	Boot flag: 0 = not active, 0x80 active
1	HD	Begin: head number
2	SEC CYL	Begin: sector and cylinder number of boot sector
1	SYS	System code: 0x83 Linux, 0x82 Linux swap etc.
1	HD	End: head number
2	SEC CYL	End: sector and cylinder number of last sector
4	low byte high byte	Relative sector number of start sector
4	low byte high byte	Number of sectors in the partition

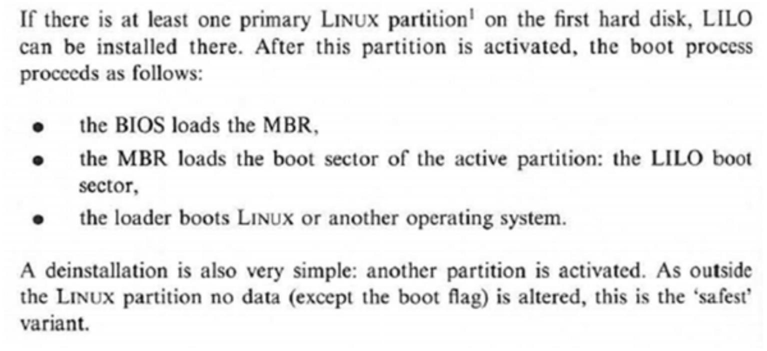
Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

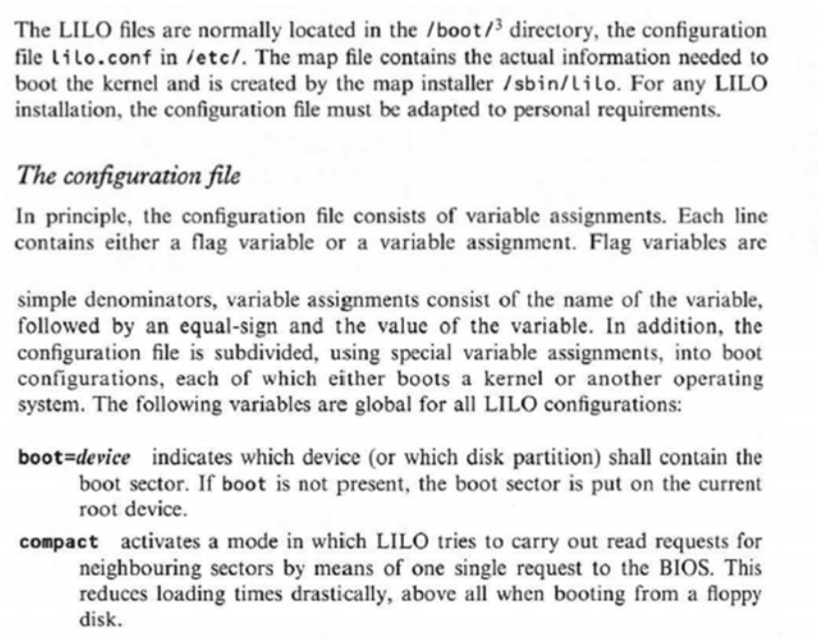


If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1



The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

boot=*device* indicates which device (or which disk partition) shall contain the boot sector. If **boot** is not present, the boot sector is put on the current root device.

compact activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

2. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

3. The method of claim 1, wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the updating comprises: removing an association of additional boot data that is associated with the boot data list from the boot data list."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein removing an association of additional boot data that is associated with the boot data list from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

4. The method of claim 1, wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the updating comprises: associating additional boot data with the boot data list; and compressing a portion of the additional boot data."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating wherein associating additional boot data with the boot data list; and compressing a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.4.1, and 2 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

5 (Preamble) A method for booting a computer system, the method comprising:	Linux Kernel, as evidenced by the example citations below, discloses "a method for booting a computer system, the method comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, booting a computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1-1.4.2 above.	

5.1 storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory;	Linux Kernel, as evidenced by the example citations below, discloses "storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing boot data in a compressed form that is associated with a portion of a boot data list in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above.	

5.2 loading the stored compressed boot data from the first memory;	Linux Kernel, as evidenced by the example citations below, discloses "loading the stored compressed boot data from the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the stored compressed boot data from the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above.	

5.3 accessing the loaded compressed boot data;	Linux Kernel, as evidenced by the example citations below, discloses "accessing the loaded compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.2 above.	

5.4 decompressing the accessed compressed boot data;	Linux Kernel, as evidenced by the example citations below, discloses "decompressing the accessed compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.3 above.	

5.5 utilizing the decompressed boot data to at least partially boot the computer system;	Linux Kernel, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to at least partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

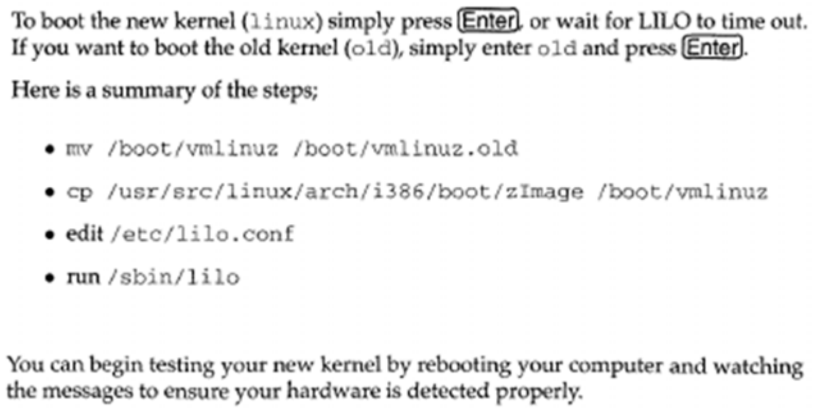
5.6 updating the boot data list;	Linux Kernel, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

5.7 wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the loading, the accessing, and the decompressing occur within a period of time which is less than a time to access the boot data from the first memory if the boot data was stored in the first memory in an uncompressed form.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1-1.3 above.	

6 (Preamble) A system comprising: a processor;	Linux Kernel, as evidenced by the example citations below, discloses "a system comprising: a processor:"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a system comprising: a processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:



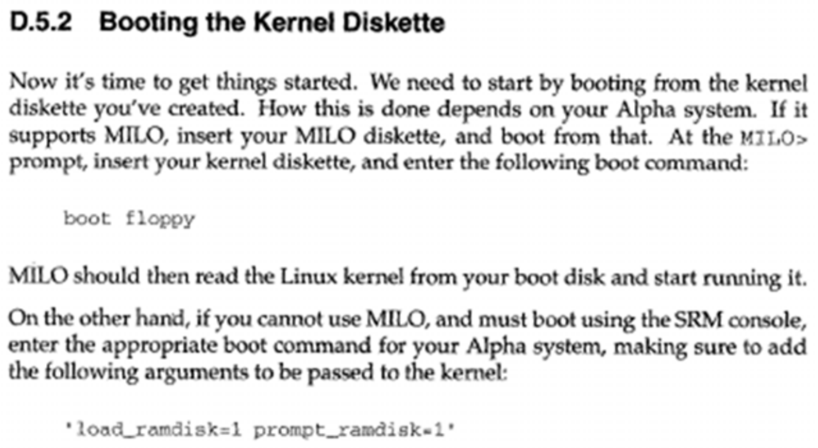
To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1



D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

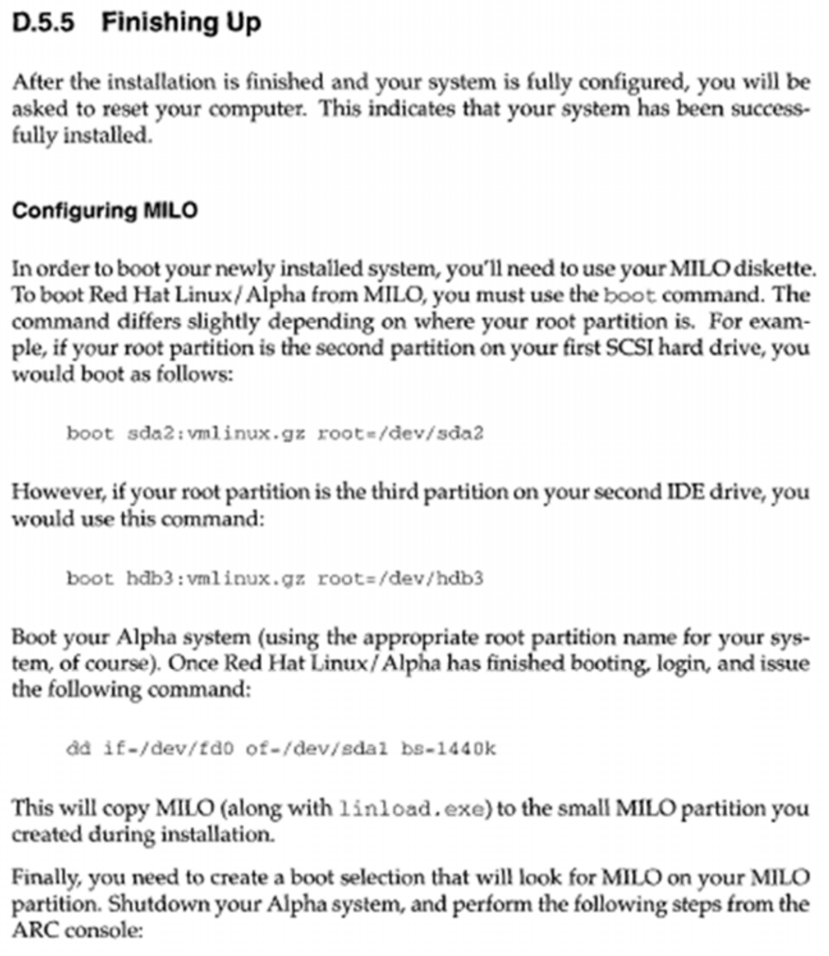
```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2



D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux / Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

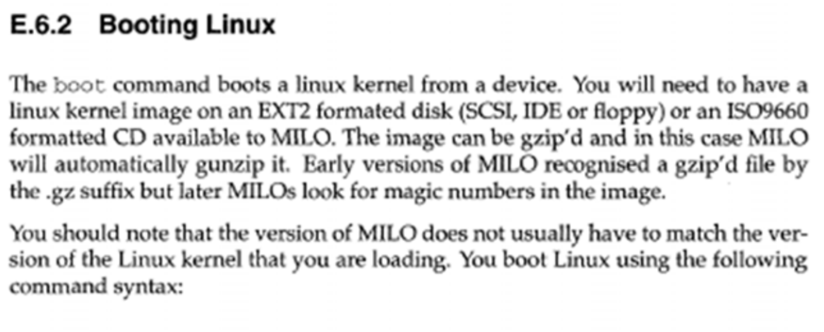
Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux / Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

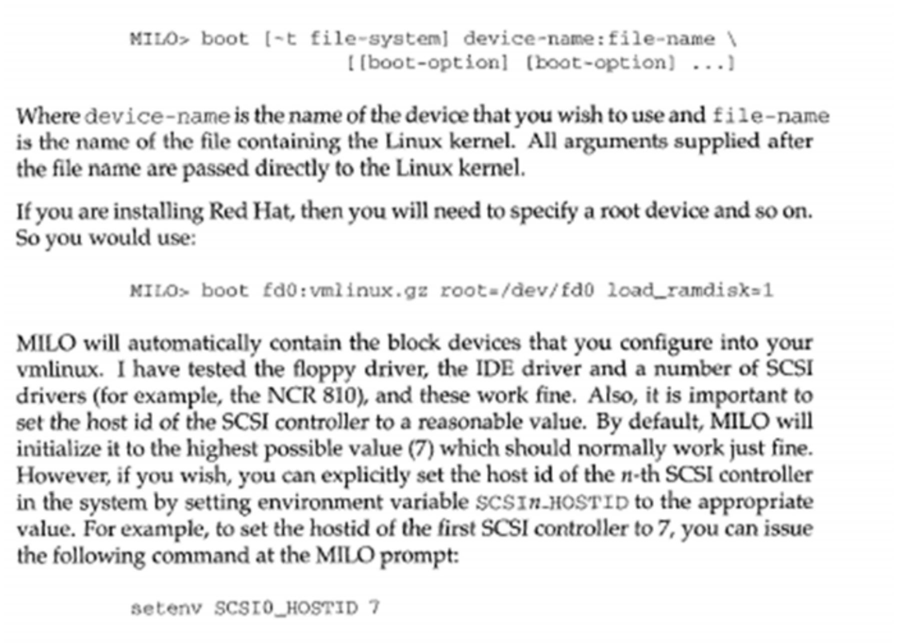
Linux Redhat, at Appendix D5.5



E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:



```
MILO> boot [-t file-system] device-name:file-name \
                 [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

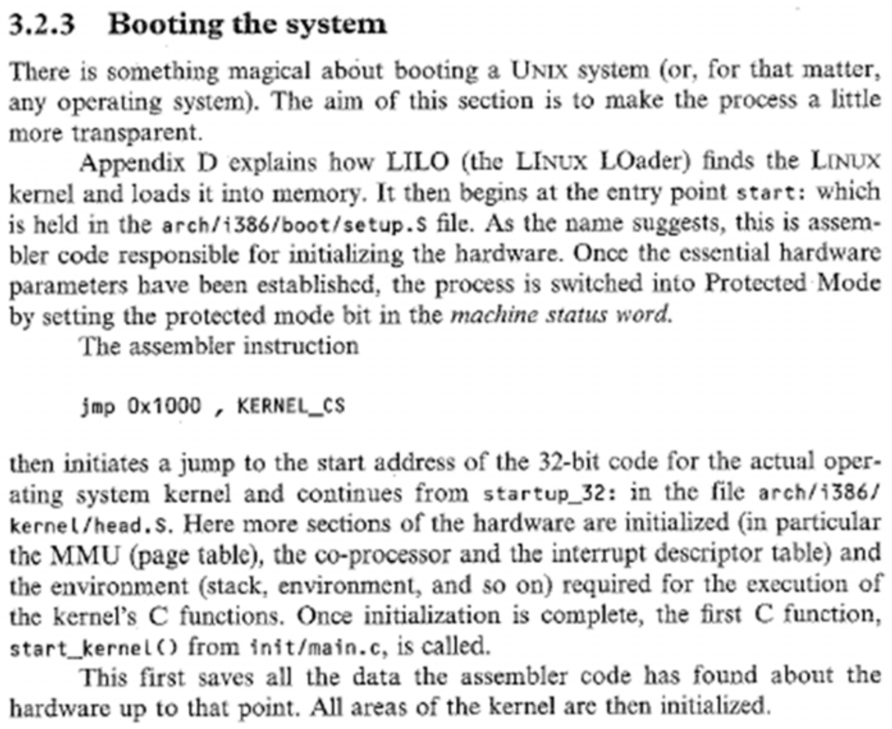
If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2



3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word*.

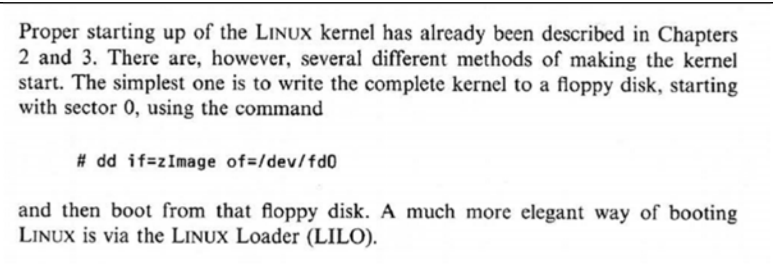
The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

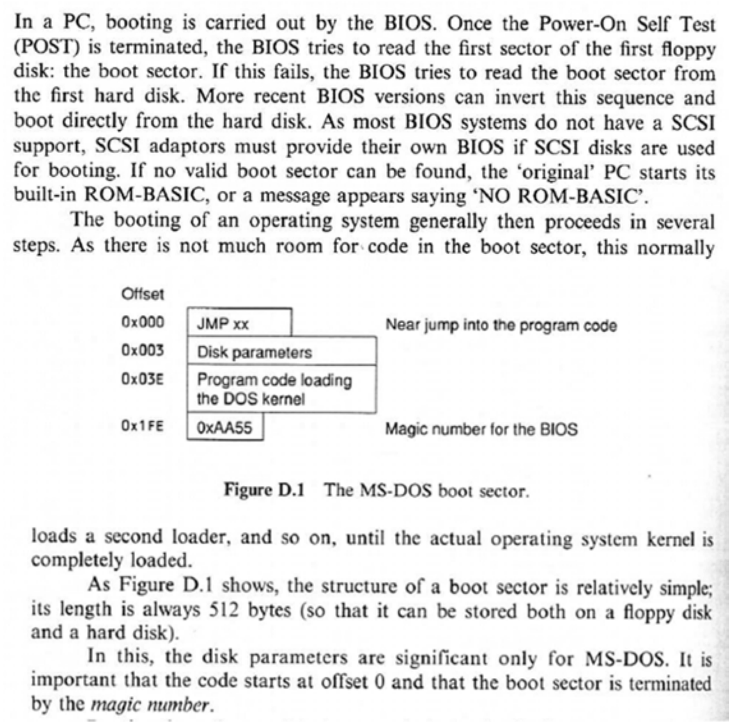


Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D



In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

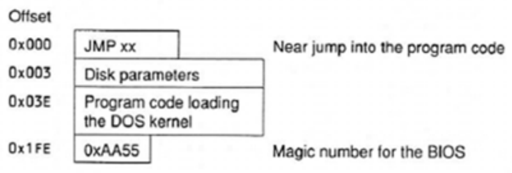

Offset
0x000
JMP xx
Near jump into the program code
0x003
Disk parameters
0x03E
Program code loading the DOS kernel
0x1FE
0xAA55
Magic number for the BIOS

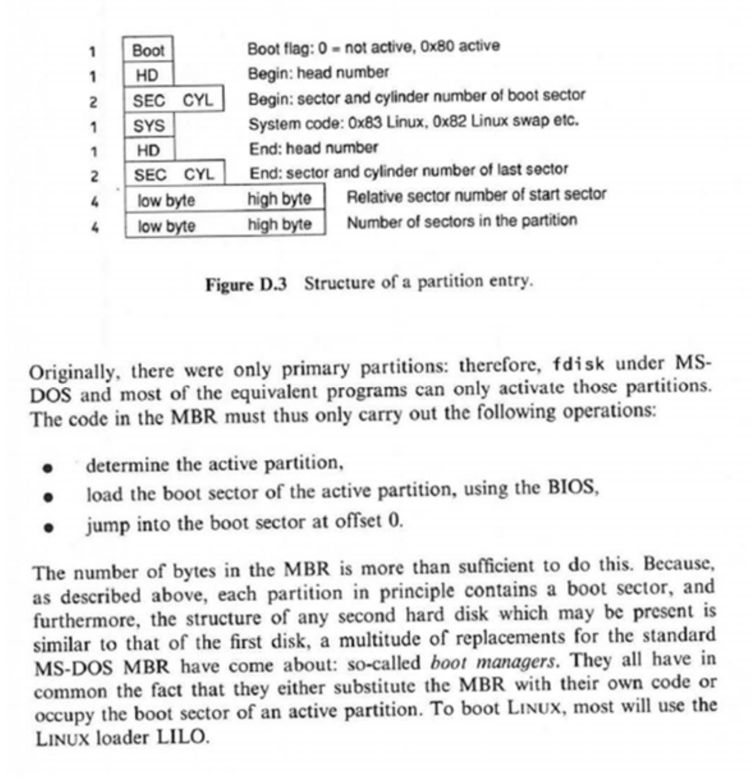
Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1



1	Boot	Boot flag: 0 = not active, 0x80 active
1	HD	Begin: head number
2	SEC CYL	Begin: sector and cylinder number of boot sector
1	SYS	System code: 0x83 Linux, 0x82 Linux swap etc.
1	HD	End: head number
2	SEC CYL	End: sector and cylinder number of last sector
4	low byte high byte	Relative sector number of start sector
4	low byte high byte	Number of sectors in the partition

Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

6.1 a memory; and	Linux Kernel, as evidenced by the example citations below, discloses “a memory”
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a memory.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple’s Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1 and 1.2 above.	

6.2 a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor	Linux Kernel, as evidenced by the example citations below, discloses "a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a second memory configured to store boot data in a compressed form for booting the system and a logic code associated with the processor), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1, 1.2, and Claim 6 (Preamble) above.	

6.3 wherein the processor is configured:	Linux Kernel, as evidenced by the example citations below, discloses "wherein the processor is configured:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the processor is configured), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 6 (Preamble) above	

6.4 to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory,	Linux Kernel, as evidenced by the example citations below, discloses "to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to load a portion of the boot data in the compressed form that is associated with a boot data list used for booting the system into the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above.	

6.5 to access the loaded portion of the boot data in the compressed form,	Linux Kernel, as evidenced by the example citations below, discloses "to access the loaded portion of the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to access the loaded portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.2 above.	

6.6 to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data, and	Linux Kernel, as evidenced by the example citations below, discloses "to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to decompress the accessed portion of the boot data in the compressed form at a rate that decreases a boot time of the system relative to booting the system with uncompressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.3 above.	

6.7 to update the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "to update the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, to update the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

8 (Preamble) A method of loading an operating system for booting a computer system, comprising:	Linux Kernel, as evidenced by the example citations below, discloses "A method of loading an operating system for booting a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method of loading an operating system for booting a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1 (Preamble) above.	

8.1 storing a portion of the operating system in a compressed form in a first memory;	Linux Kernel, as evidenced by the example citations below, discloses "storing a portion of the operating system in a compressed form in a first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing a portion of the operating system in a compressed form in a first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 6.2 above.	

8.2 loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list;	Linux Kernel, as evidenced by the example citations below, discloses "loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the portion of the operating system from the first memory to a second memory, the portion of the operating system being associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above.	

8.3 accessing the loaded portion of the operating system from the second memory in the compressed form;	Linux Kernel, as evidenced by the example citations below, discloses "accessing the loaded portion of the operating system from the second memory in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded portion of the operating system from the second memory in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.2 above.	

8.4 decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system;	Linux Kernel, as evidenced by the example citations below, discloses "decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed portion of the operating system to provide a decompressed portion of the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.5 utilizing the decompressed portion of the operating system to at least partially boot the computer system; and	Linux Kernel, as evidenced by the example citations below, discloses "utilizing the decompressed portion of the operating system to at least partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed portion of the operating system to at least partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.3 and 1.4.2 above.	

8.6 updating the boot data list,	Linux Kernel, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

8.7 wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the portion of the operating system is accessed and decompressed at a rate that is faster than accessing the loaded portion of the operating system from the first memory if the portion of the operating system was to be stored in the first memory in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1-1.3 and 5.7 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

9.1 The method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list; and	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 8, further comprising: compressing an additional portion of the operating system that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, compressing an additional portion of the operating system that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above.	

9.2 storing the additional portion of the operating system in the first memory, and	Linux Kernel, as evidenced by the example citations below, discloses "storing the additional portion of the operating system in the first memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, storing the additional portion of the operating system in the first memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 8.1 above.	

9.3 wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system.	Linux Kernel, as evidenced by the example citations below, discloses "wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the utilizing comprises: utilizing the stored additional portion of the operating system to at least further partially boot the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 8.5 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

10. The method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 9, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, wherein the compressing comprises: compressing the additional portion of the operating system with a data compression encoder), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 9.1 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

11 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Linux Kernel, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1 (Preamble) above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

11.1 loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system;	Linux Kernel, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device into a memory upon initialization of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 and 6 (Preamble) above.	

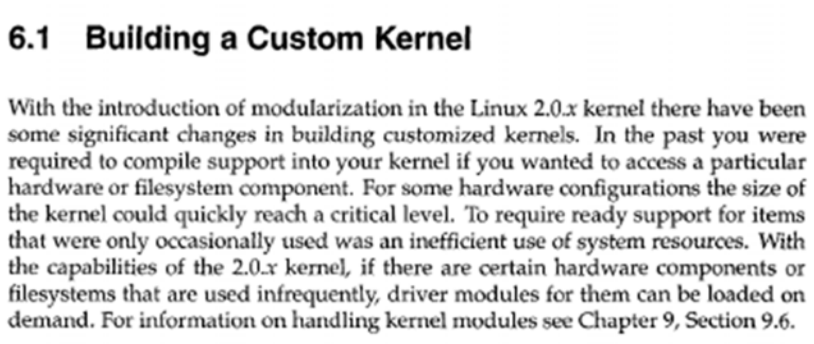
11.2 accessing the loaded boot data in compressed form from the memory;	Linux Kernel, as evidenced by the example citations below, discloses "accessing the loaded boot data in compressed form from the memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in compressed form from the memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.2 above.	

11.3 decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Linux Kernel, as evidenced by the example citations below, discloses "decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.3 above.	

11.4 utilizing the decompressed boot data to load at least a portion of the operating system for the computer system; and	Linux Kernel, as evidenced by the example citations below, discloses "utilizing the decompressed boot data to load at least a portion of the operating system for the computer system"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, utilizing the decompressed boot data to load at least a portion of the operating system for the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

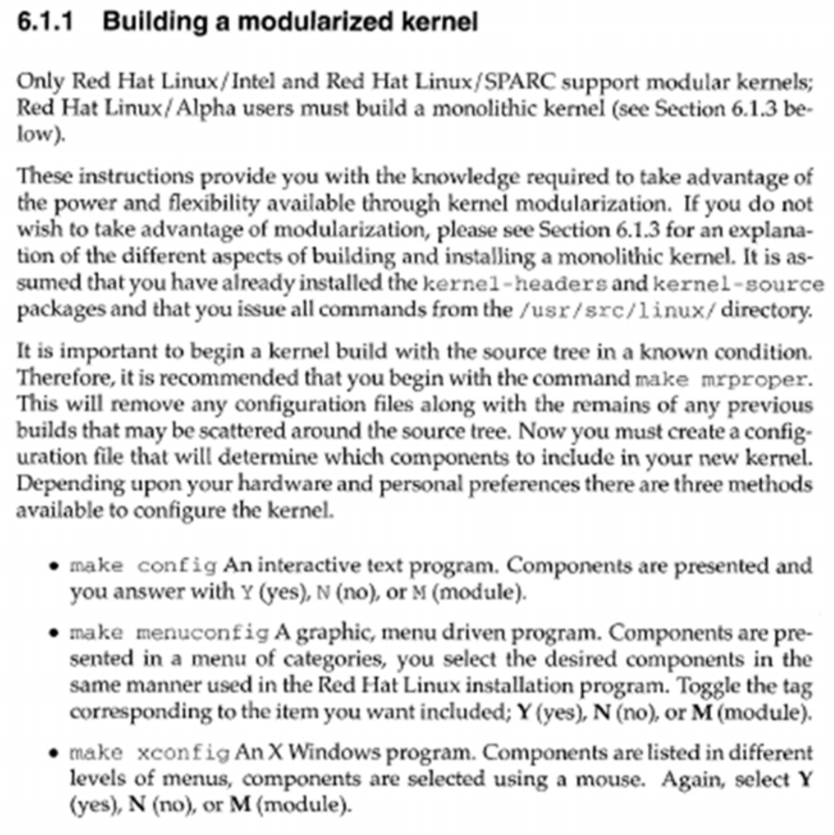
Linux Kernel discloses this limitation:



6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.*x* kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.*x* kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

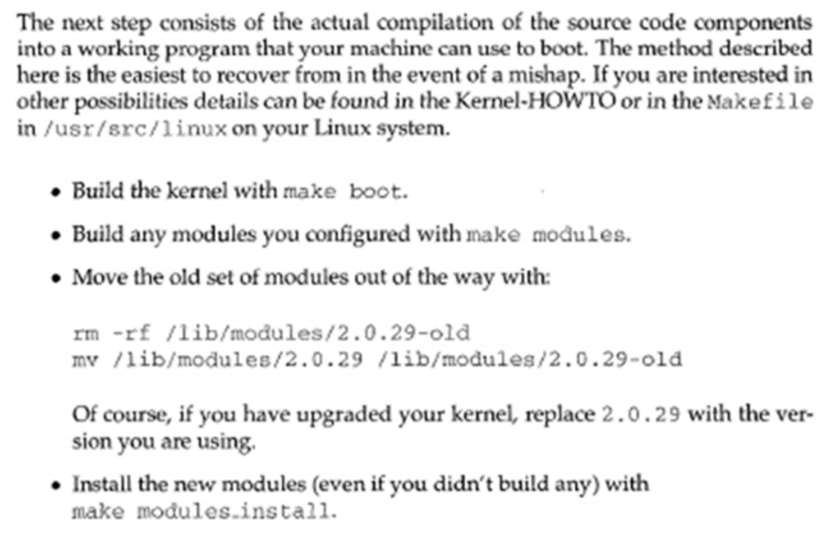
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1



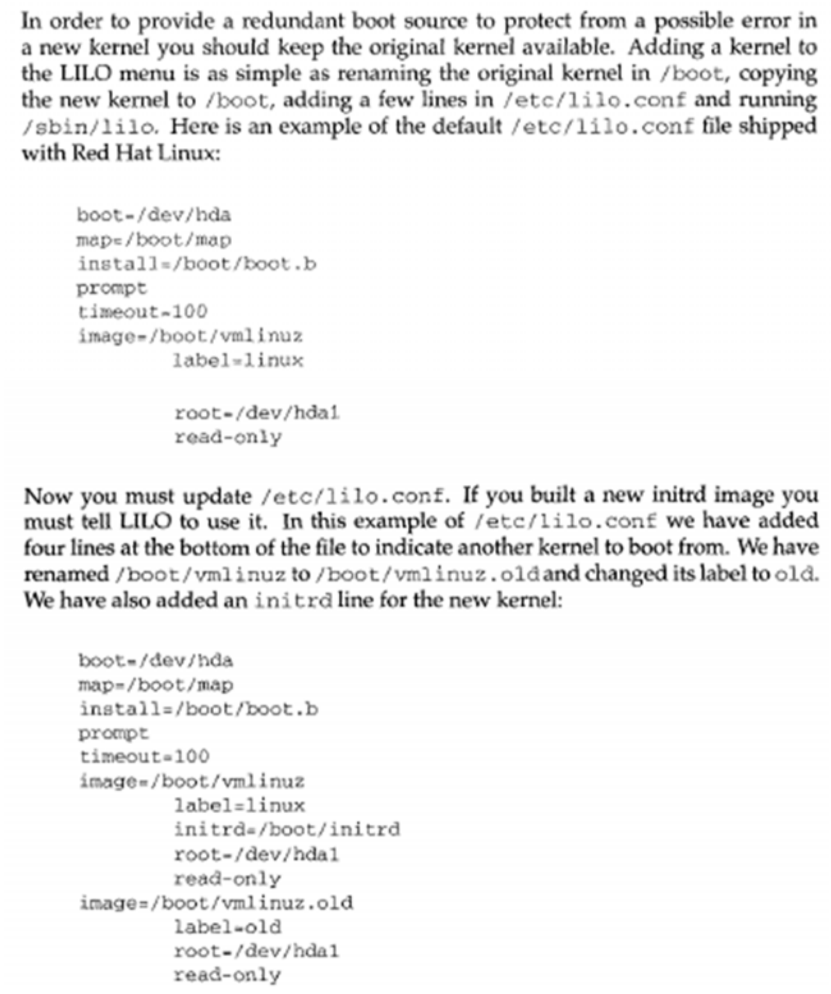
The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

  ```
  rm -rf /lib/modules/2.0.29-old
  mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
  ```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1



In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in /boot, copying the new kernel to /boot, adding a few lines in /etc/lilo.conf and running /sbin/lilo. Here is an example of the default /etc/lilo.conf file shipped with Red Hat Linux:

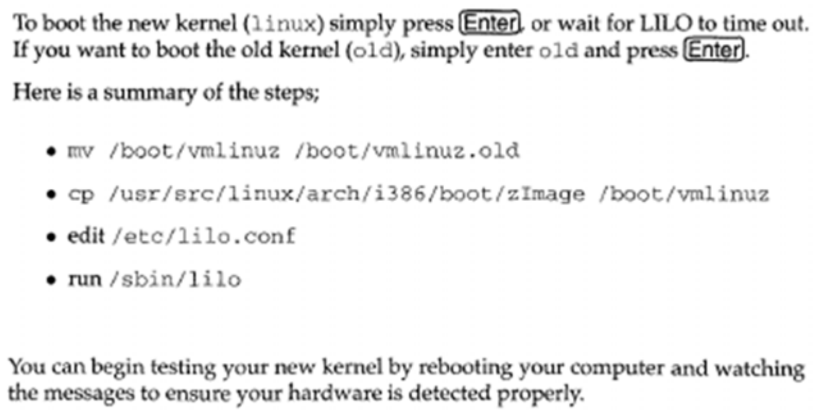
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update /etc/lilo.conf. If you built a new initrd image you must tell LILO to use it. In this example of /etc/lilo.conf we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed /boot/vmlinuz to /boot/vmlinuz.old and changed its label to old. We have also added an initrd line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1



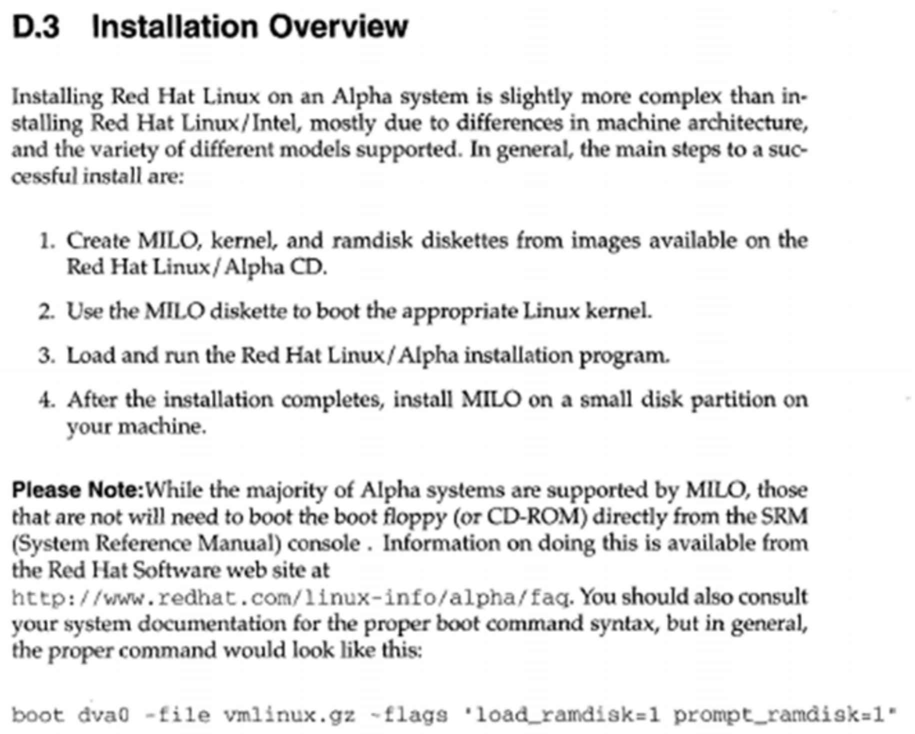
To boot the new kernel (linux) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (old), simply enter old and press Enter.

Here is a summary of the steps;

- mv /boot/vmlinuz /boot/vmlinuz.old
- cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz
- edit /etc/lilo.conf
- run /sbin/lilo

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1



D.3 Installation Overview

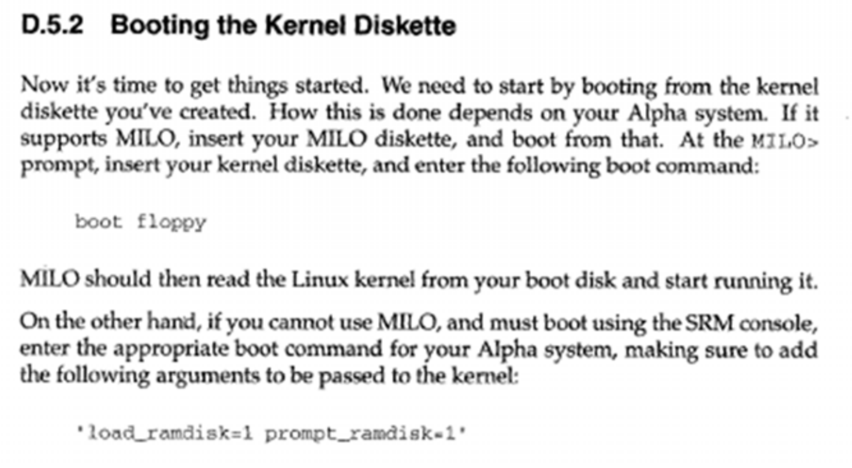
Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3



D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

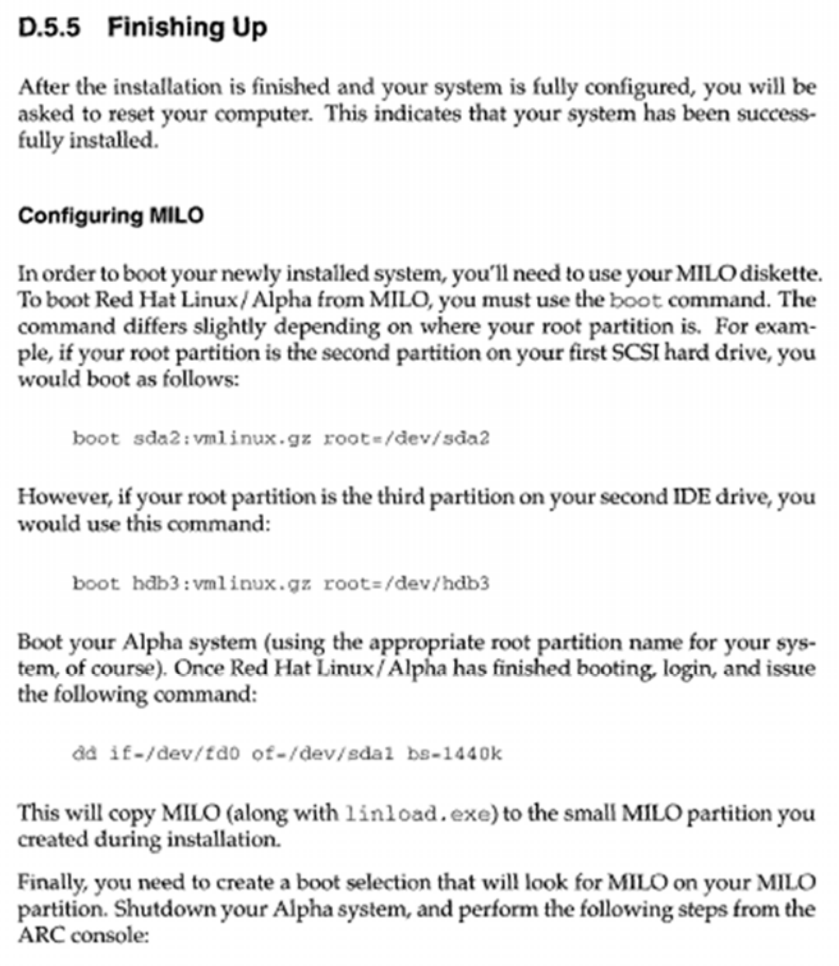
```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2



D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

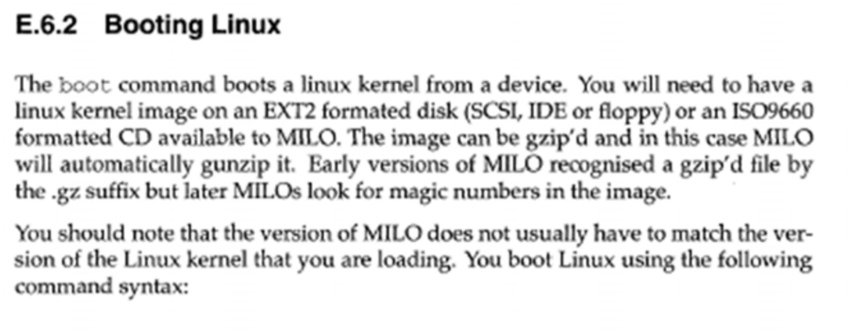
Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

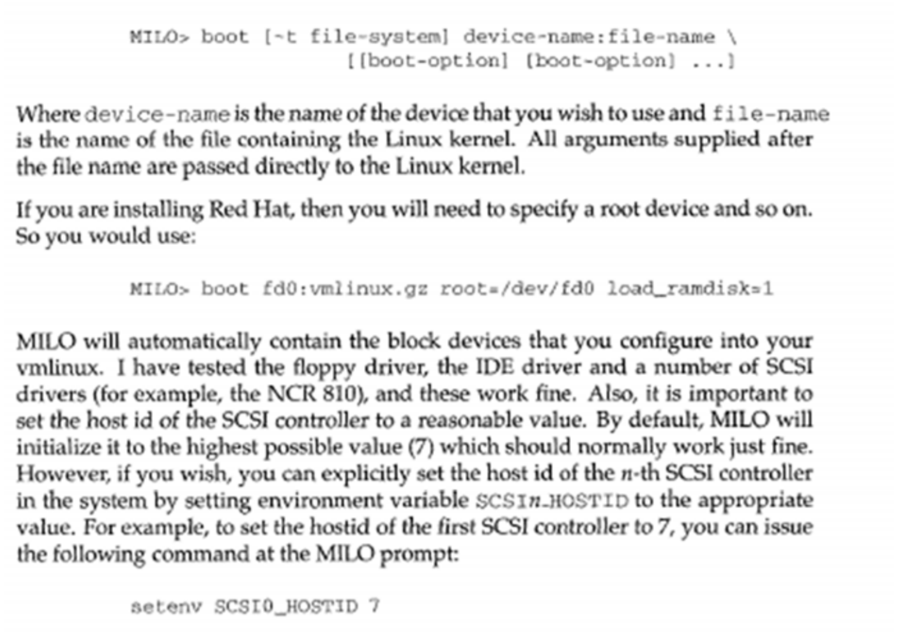
Linux Redhat, at Appendix D5.5



E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:



```
MILO> boot [-t file-system] device-name:file-name \
                 [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

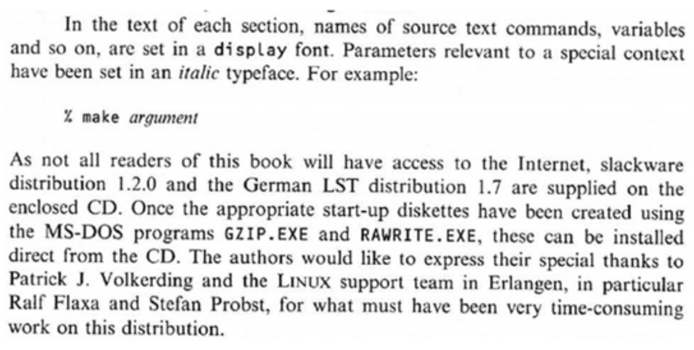
If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

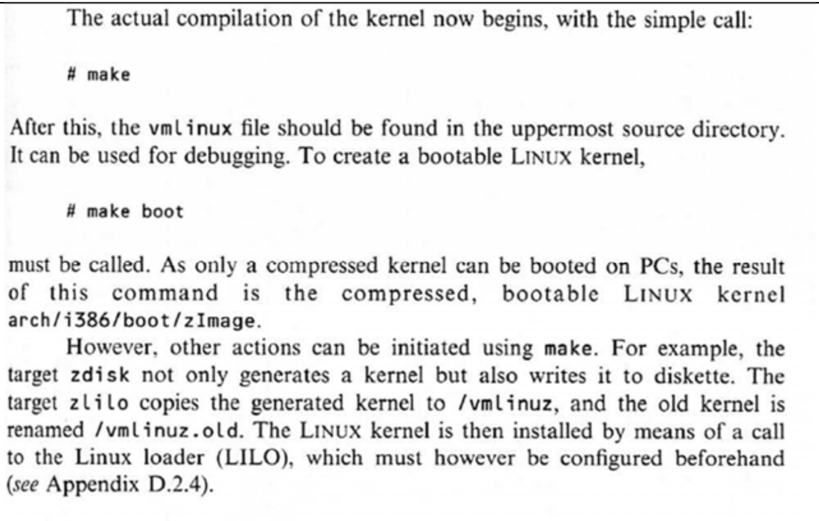


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface



The actual compilation of the kernel now begins, with the simple call:

```
# make
```

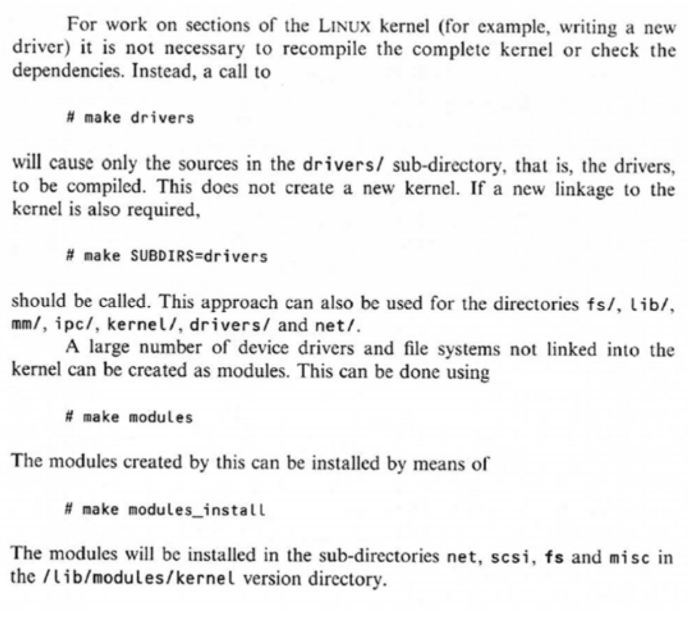
After this, the vmlinux file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel arch/i386/boot/zImage.

However, other actions can be initiated using make. For example, the target zdisk not only generates a kernel but also writes it to diskette. The target zlilo copies the generated kernel to /vmlinuz, and the old kernel is renamed /vmlinuz.old. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2



For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the drivers/ sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories fs/, lib/, mm/, ipc/, kernel/, drivers/ and net/.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

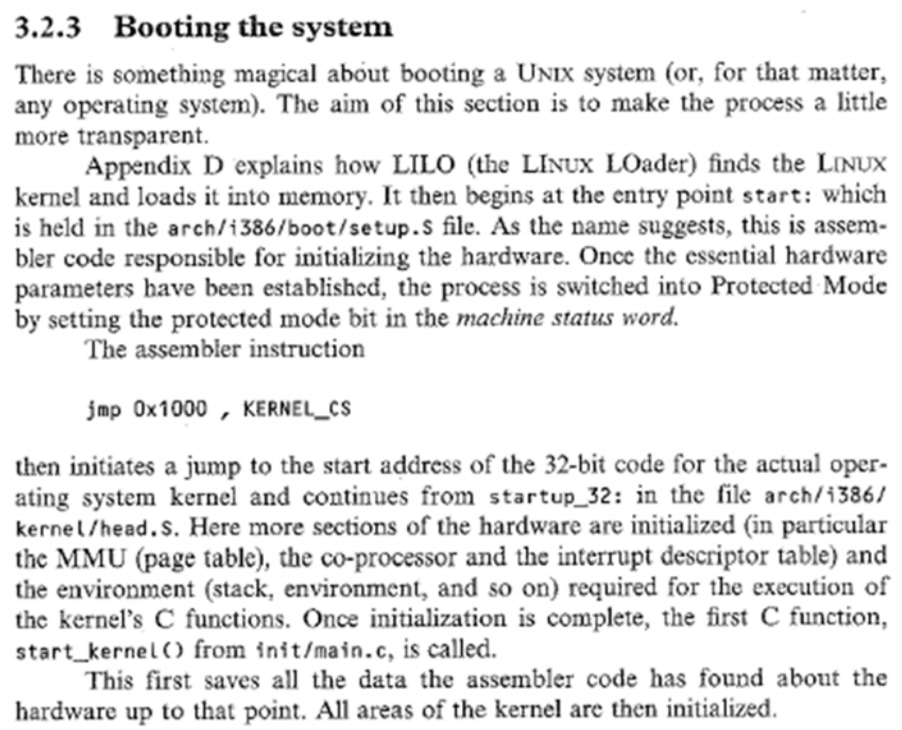
```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories net, scsi, fs and misc in the /lib/modules/kernel version directory.

Beck, at 2.2



3.2.3 Booting the system

There is something magical about booting a UNIX system (or, for that matter, any operating system). The aim of this section is to make the process a little more transparent.

Appendix D explains how LILO (the LINUX LOader) finds the LINUX kernel and loads it into memory. It then begins at the entry point `start:` which is held in the `arch/i386/boot/setup.S` file. As the name suggests, this is assembler code responsible for initializing the hardware. Once the essential hardware parameters have been established, the process is switched into Protected Mode by setting the protected mode bit in the *machine status word.*

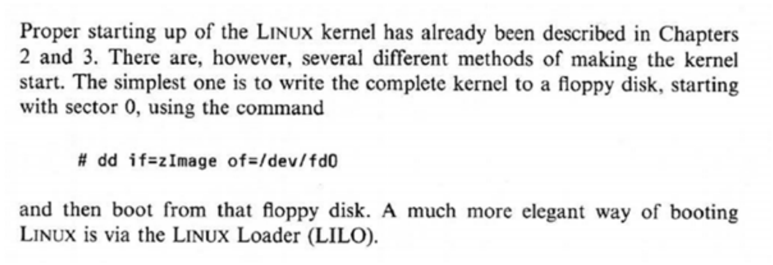
The assembler instruction

```
jmp 0x1000 , KERNEL_CS
```

then initiates a jump to the start address of the 32-bit code for the actual operating system kernel and continues from `startup_32:` in the file `arch/i386/kernel/head.S`. Here more sections of the hardware are initialized (in particular the MMU (page table), the co-processor and the interrupt descriptor table) and the environment (stack, environment, and so on) required for the execution of the kernel's C functions. Once initialization is complete, the first C function, `start_kernel()` from `init/main.c`, is called.

This first saves all the data the assembler code has found about the hardware up to that point. All areas of the kernel are then initialized.

Beck, at 3.2.3

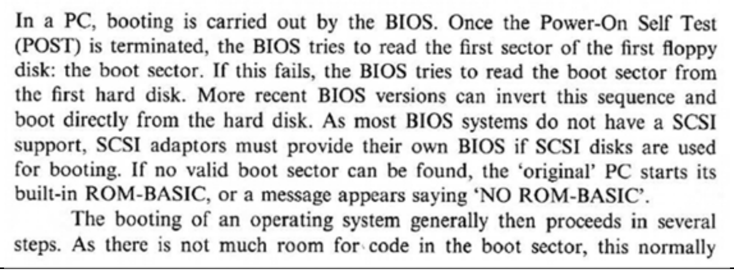


Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D



In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

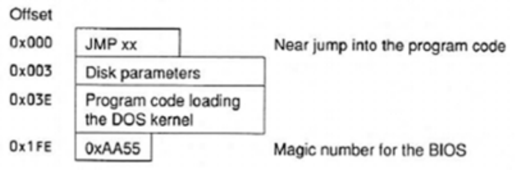

Offset
0x000
JMP xx
Near jump into the program code
0x003
Disk parameters
0x03E
Program code loading the DOS kernel
0x1FE
0xAA55
Magic number for the BIOS

Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1

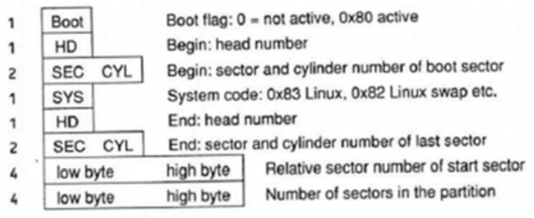

1
Boot
Boot flag: 0 = not active, 0x80 active
1
HD
Begin: head number
2
SEC CYL
Begin: sector and cylinder number of boot sector
1
SYS
System code: 0x83 Linux, 0x82 Linux swap etc.
1
HD
End: head number
2
SEC CYL
End: sector and cylinder number of last sector
4
low byte high byte
Relative sector number of start sector
4
low byte high byte
Number of sectors in the partition

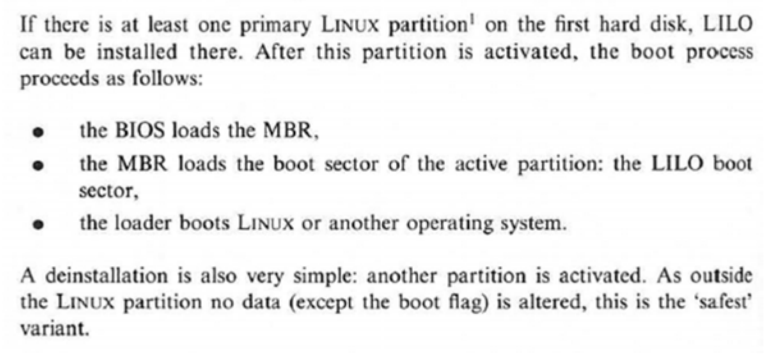
Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

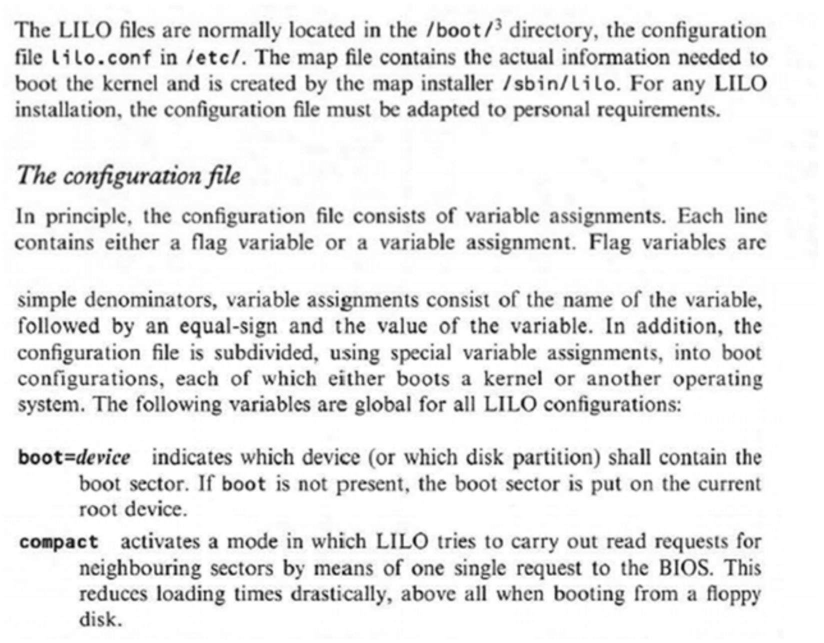


If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

Beck, at Appendix D.2.1



The LILO files are normally located in the **/boot/**[3] directory, the configuration file **lilo.conf** in **/etc/**. The map file contains the actual information needed to boot the kernel and is created by the map installer **/sbin/lilo**. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are

simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

boot=*device* indicates which device (or which disk partition) shall contain the boot sector. If **boot** is not present, the boot sector is put on the current root device.

compact activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

11.5 updating the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

13 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Linux Kernel, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1 (Preamble) above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

13.1 loading boot data in a compressed form that is associated with a boot data list from a boot device;	Linux Kernel, as evidenced by the example citations below, discloses "loading boot data in a compressed form that is associated with a boot data list from a boot device"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading boot data in a compressed form that is associated with a boot data list from a boot device), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

13.2 accessing the loaded boot data in the compressed form;	Linux Kernel, as evidenced by the example citations below, discloses "accessing the loaded boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing the loaded boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.2 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

13.3 decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form;	Linux Kernel, as evidenced by the example citations below, discloses "decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, decompressing the accessed boot data in the compressed form at a rate that decreases a time to load the operating system relative to loading the operating system with the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.3 above.	

13.4 updating the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

14 (Preamble) A method for providing accelerated loading of an operating system in a computer system, comprising:	Linux Kernel, as evidenced by the example citations below, discloses "a method for providing accelerated loading of an operating system in a computer system, comprising"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, a method for providing accelerated loading of an operating system in a computer system, comprising), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1 (Preamble) above.	

14.1 accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list;	Linux Kernel, as evidenced by the example citations below, discloses "accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, accessing boot data for booting the computer system, wherein a portion of the boot data is in a compressed form and is associated with a boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1 and 1.2 above.	

14.2 loading the boot data into a memory; and	Linux Kernel, as evidenced by the example citations below, discloses "loading the boot data into a memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, loading the boot data into a memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.1 above.	

14.3 servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form; and	Linux Kernel, as evidenced by the example citations below, discloses "servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, servicing a request for the boot data from the computer system to access the loaded compressed boot data and to decompress the accessed compressed boot data at a rate that decreases a boot time of the operating system relative to loading the operating system utilizing the boot data in an uncompressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.2 and 1.3 above.	

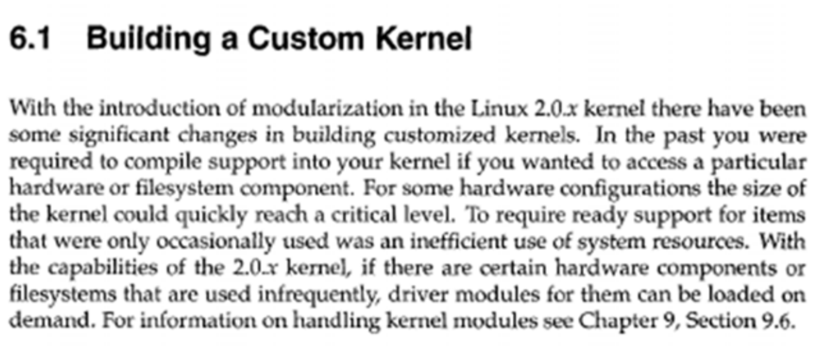
14.4 updating the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "updating the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, updating the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1.4.1 above.	

15. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1 (Preamble) and 1.1 above.	

16. The method of claim 14, wherein the operating system comprises: a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 14, wherein the operating system comprises: a plurality of files."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

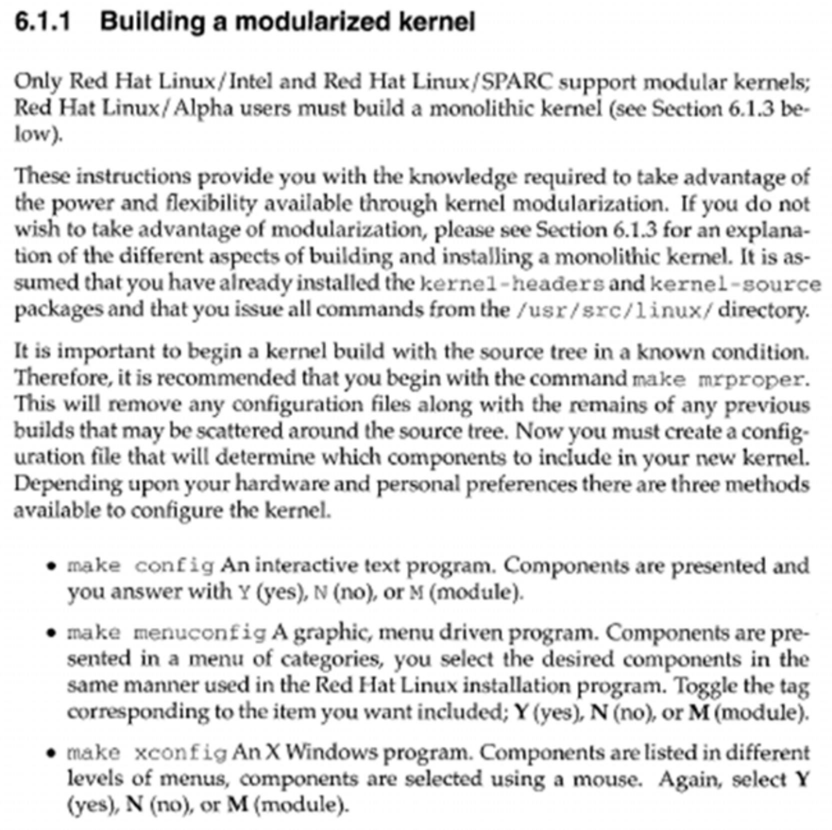
Linux Kernel discloses this limitation:



6.1 Building a Custom Kernel

With the introduction of modularization in the Linux 2.0.x kernel there have been some significant changes in building customized kernels. In the past you were required to compile support into your kernel if you wanted to access a particular hardware or filesystem component. For some hardware configurations the size of the kernel could quickly reach a critical level. To require ready support for items that were only occasionally used was an inefficient use of system resources. With the capabilities of the 2.0.x kernel, if there are certain hardware components or filesystems that are used infrequently, driver modules for them can be loaded on demand. For information on handling kernel modules see Chapter 9, Section 9.6.

Linux Redhat, at 6.1



6.1.1 Building a modularized kernel

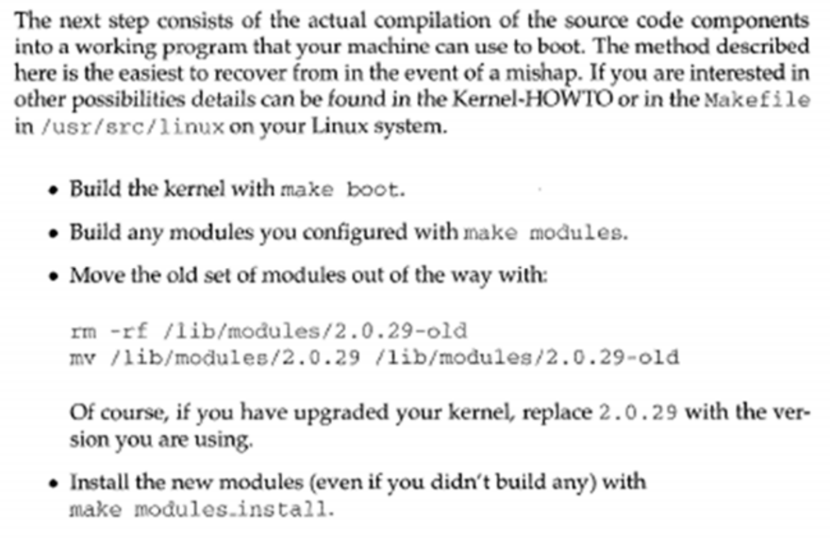
Only Red Hat Linux/Intel and Red Hat Linux/SPARC support modular kernels; Red Hat Linux/Alpha users must build a monolithic kernel (see Section 6.1.3 below).

These instructions provide you with the knowledge required to take advantage of the power and flexibility available through kernel modularization. If you do not wish to take advantage of modularization, please see Section 6.1.3 for an explanation of the different aspects of building and installing a monolithic kernel. It is assumed that you have already installed the `kernel-headers` and `kernel-source` packages and that you issue all commands from the `/usr/src/linux/` directory.

It is important to begin a kernel build with the source tree in a known condition. Therefore, it is recommended that you begin with the command `make mrproper`. This will remove any configuration files along with the remains of any previous builds that may be scattered around the source tree. Now you must create a configuration file that will determine which components to include in your new kernel. Depending upon your hardware and personal preferences there are three methods available to configure the kernel.

- `make config` An interactive text program. Components are presented and you answer with `Y` (yes), `N` (no), or `M` (module).
- `make menuconfig` A graphic, menu driven program. Components are presented in a menu of categories, you select the desired components in the same manner used in the Red Hat Linux installation program. Toggle the tag corresponding to the item you want included; **Y** (yes), **N** (no), or **M** (module).
- `make xconfig` An X Windows program. Components are listed in different levels of menus, components are selected using a mouse. Again, select **Y** (yes), **N** (no), or **M** (module).

Linux Redhat, at 6.1.1



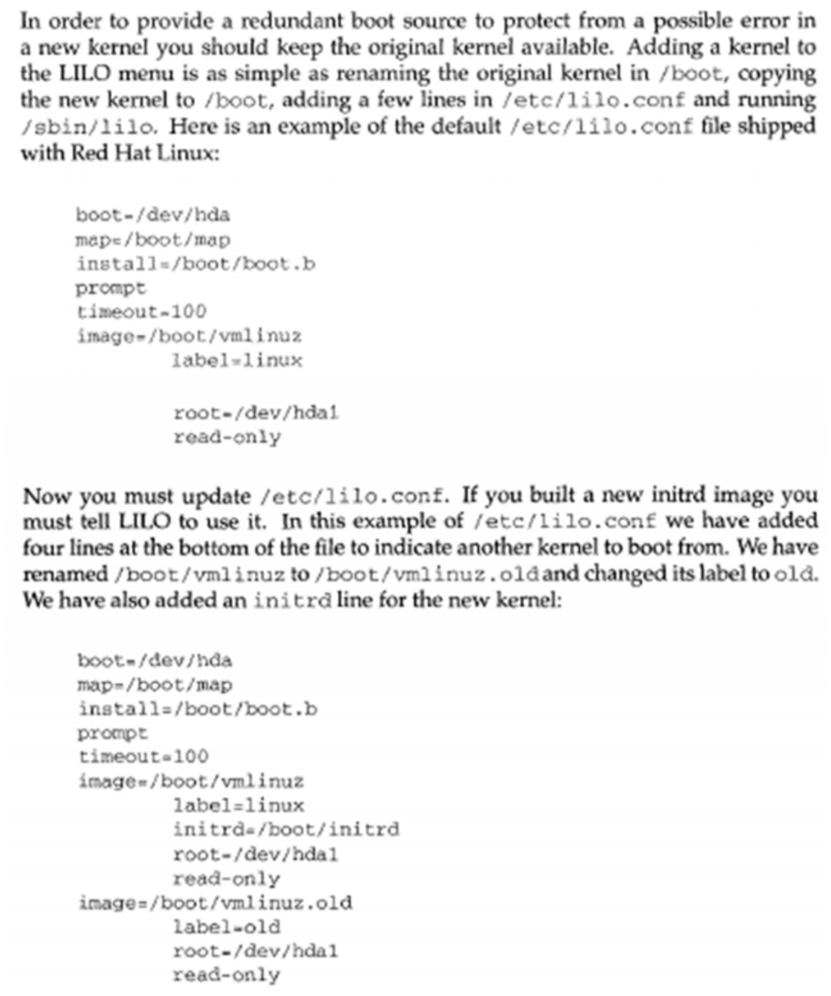
The next step consists of the actual compilation of the source code components into a working program that your machine can use to boot. The method described here is the easiest to recover from in the event of a mishap. If you are interested in other possibilities details can be found in the Kernel-HOWTO or in the `Makefile` in `/usr/src/linux` on your Linux system.

- Build the kernel with `make boot`.
- Build any modules you configured with `make modules`.
- Move the old set of modules out of the way with:

```
rm -rf /lib/modules/2.0.29-old
mv /lib/modules/2.0.29 /lib/modules/2.0.29-old
```

  Of course, if you have upgraded your kernel, replace `2.0.29` with the version you are using.
- Install the new modules (even if you didn't build any) with `make modules_install`.

Linux Redhat, at 6.1.1



In order to provide a redundant boot source to protect from a possible error in a new kernel you should keep the original kernel available. Adding a kernel to the LILO menu is as simple as renaming the original kernel in `/boot`, copying the new kernel to `/boot`, adding a few lines in `/etc/lilo.conf` and running `/sbin/lilo`. Here is an example of the default `/etc/lilo.conf` file shipped with Red Hat Linux:

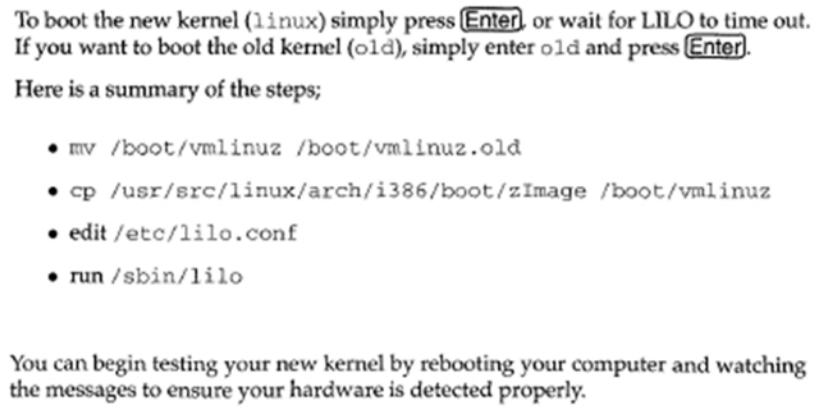
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux

        root=/dev/hda1
        read-only
```

Now you must update `/etc/lilo.conf`. If you built a new initrd image you must tell LILO to use it. In this example of `/etc/lilo.conf` we have added four lines at the bottom of the file to indicate another kernel to boot from. We have renamed `/boot/vmlinuz` to `/boot/vmlinuz.old` and changed its label to `old`. We have also added an `initrd` line for the new kernel:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
image=/boot/vmlinuz
        label=linux
        initrd=/boot/initrd
        root=/dev/hda1
        read-only
image=/boot/vmlinuz.old
        label=old
        root=/dev/hda1
        read-only
```

Linux Redhat, at 6.1.1



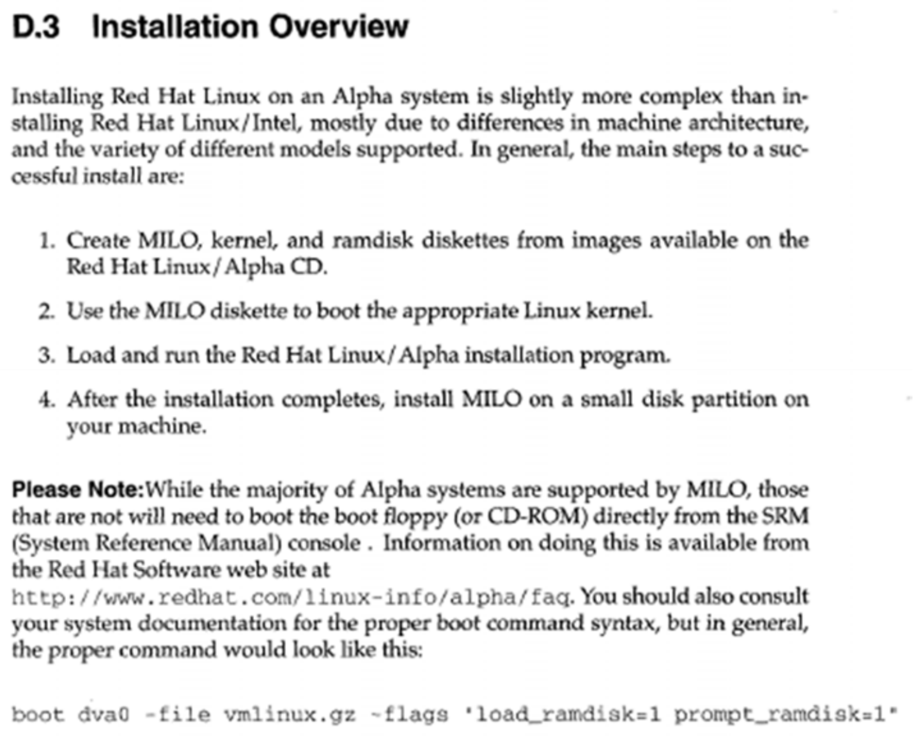
To boot the new kernel (`linux`) simply press Enter, or wait for LILO to time out. If you want to boot the old kernel (`old`), simply enter `old` and press Enter.

Here is a summary of the steps;

- `mv /boot/vmlinuz /boot/vmlinuz.old`
- `cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz`
- edit `/etc/lilo.conf`
- run `/sbin/lilo`

You can begin testing your new kernel by rebooting your computer and watching the messages to ensure your hardware is detected properly.

Linux Redhat, at 6.1.1



D.3 Installation Overview

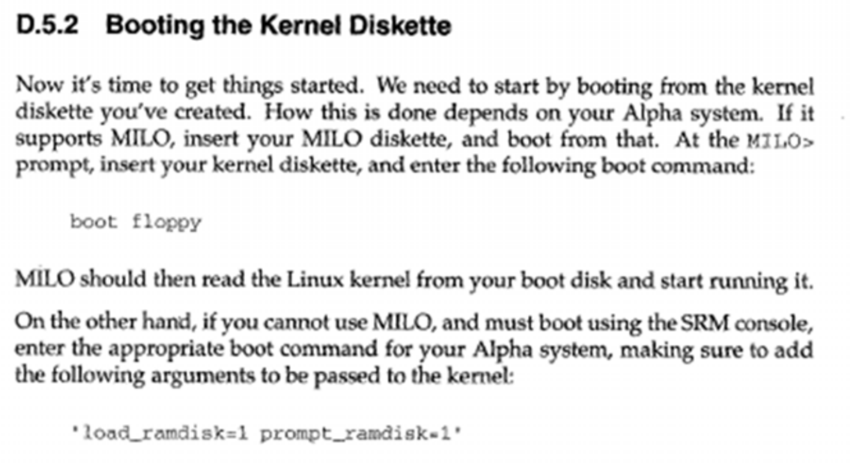
Installing Red Hat Linux on an Alpha system is slightly more complex than installing Red Hat Linux/Intel, mostly due to differences in machine architecture, and the variety of different models supported. In general, the main steps to a successful install are:

1. Create MILO, kernel, and ramdisk diskettes from images available on the Red Hat Linux/Alpha CD.
2. Use the MILO diskette to boot the appropriate Linux kernel.
3. Load and run the Red Hat Linux/Alpha installation program.
4. After the installation completes, install MILO on a small disk partition on your machine.

Please Note: While the majority of Alpha systems are supported by MILO, those that are not will need to boot the boot floppy (or CD-ROM) directly from the SRM (System Reference Manual) console . Information on doing this is available from the Red Hat Software web site at `http://www.redhat.com/linux-info/alpha/faq`. You should also consult your system documentation for the proper boot command syntax, but in general, the proper command would look like this:

```
boot dva0 -file vmlinux.gz -flags 'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.3



D.5.2 Booting the Kernel Diskette

Now it's time to get things started. We need to start by booting from the kernel diskette you've created. How this is done depends on your Alpha system. If it supports MILO, insert your MILO diskette, and boot from that. At the `MILO>` prompt, insert your kernel diskette, and enter the following boot command:

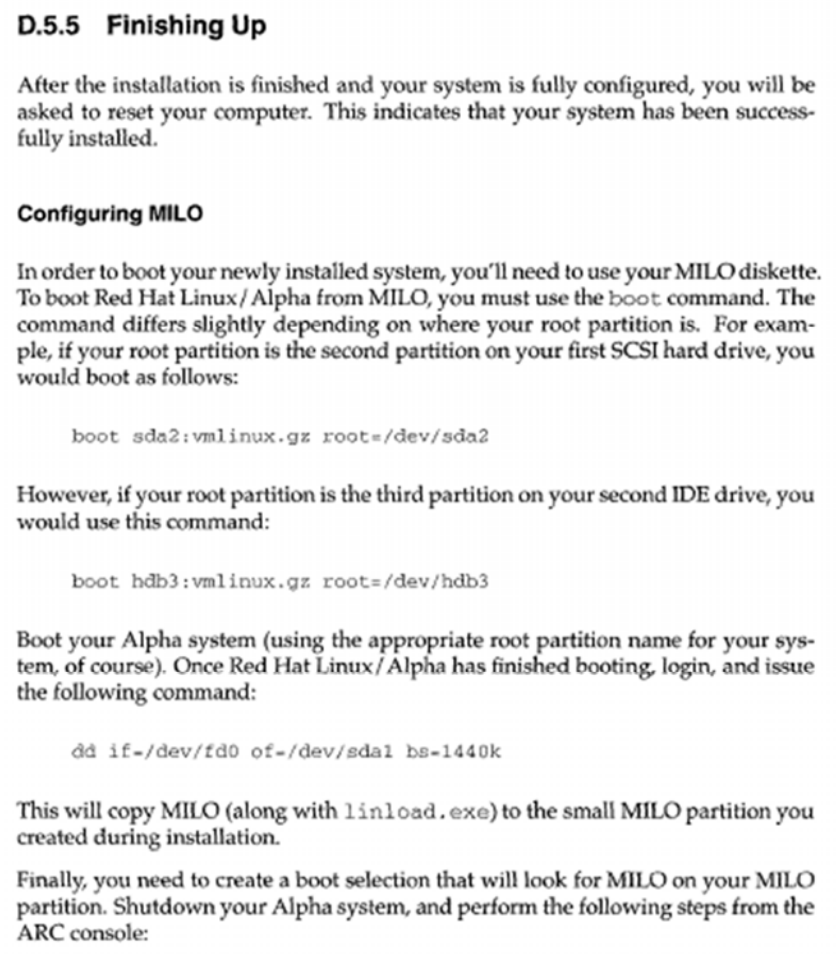
```
boot floppy
```

MILO should then read the Linux kernel from your boot disk and start running it.

On the other hand, if you cannot use MILO, and must boot using the SRM console, enter the appropriate boot command for your Alpha system, making sure to add the following arguments to be passed to the kernel:

```
'load_ramdisk=1 prompt_ramdisk=1'
```

Linux Redhat, at Appendix D.5.2



D.5.5 Finishing Up

After the installation is finished and your system is fully configured, you will be asked to reset your computer. This indicates that your system has been successfully installed.

Configuring MILO

In order to boot your newly installed system, you'll need to use your MILO diskette. To boot Red Hat Linux/Alpha from MILO, you must use the `boot` command. The command differs slightly depending on where your root partition is. For example, if your root partition is the second partition on your first SCSI hard drive, you would boot as follows:

```
boot sda2:vmlinux.gz root=/dev/sda2
```

However, if your root partition is the third partition on your second IDE drive, you would use this command:

```
boot hdb3:vmlinux.gz root=/dev/hdb3
```

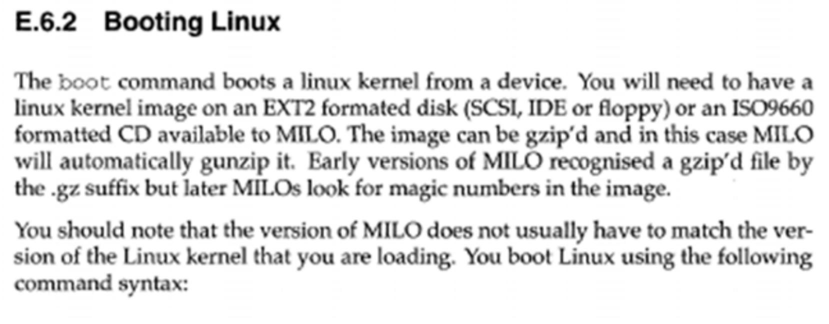
Boot your Alpha system (using the appropriate root partition name for your system, of course). Once Red Hat Linux/Alpha has finished booting, login, and issue the following command:

```
dd if=/dev/fd0 of=/dev/sda1 bs=1440k
```

This will copy MILO (along with `linload.exe`) to the small MILO partition you created during installation.

Finally, you need to create a boot selection that will look for MILO on your MILO partition. Shutdown your Alpha system, and perform the following steps from the ARC console:

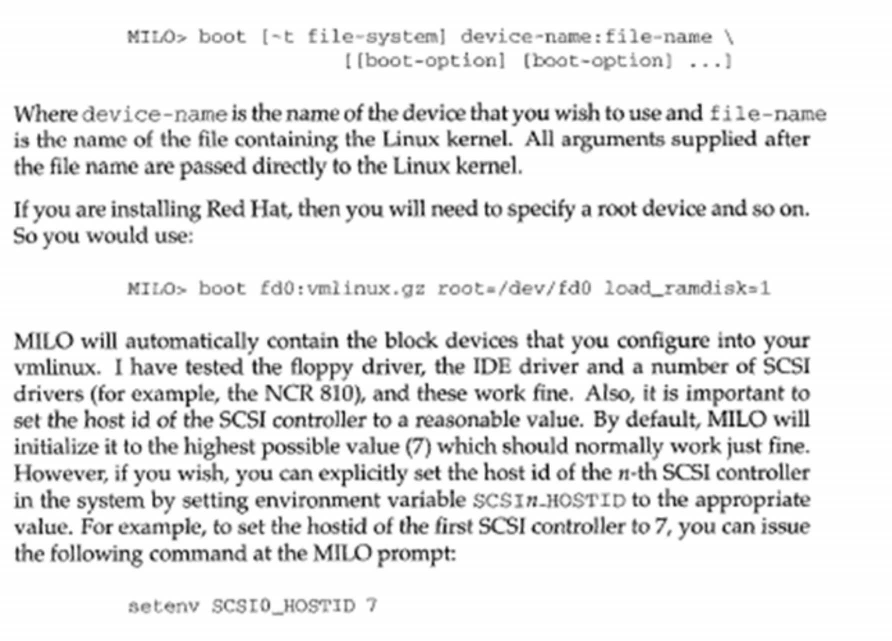
Linux Redhat, at Appendix D5.5



E.6.2 Booting Linux

The `boot` command boots a linux kernel from a device. You will need to have a linux kernel image on an EXT2 formated disk (SCSI, IDE or floppy) or an ISO9660 formatted CD available to MILO. The image can be gzip'd and in this case MILO will automatically gunzip it. Early versions of MILO recognised a gzip'd file by the .gz suffix but later MILOs look for magic numbers in the image.

You should note that the version of MILO does not usually have to match the version of the Linux kernel that you are loading. You boot Linux using the following command syntax:



```
MILO> boot [-t file-system] device-name:file-name \
                   [[boot-option] [boot-option] ...]
```

Where `device-name` is the name of the device that you wish to use and `file-name` is the name of the file containing the Linux kernel. All arguments supplied after the file name are passed directly to the Linux kernel.

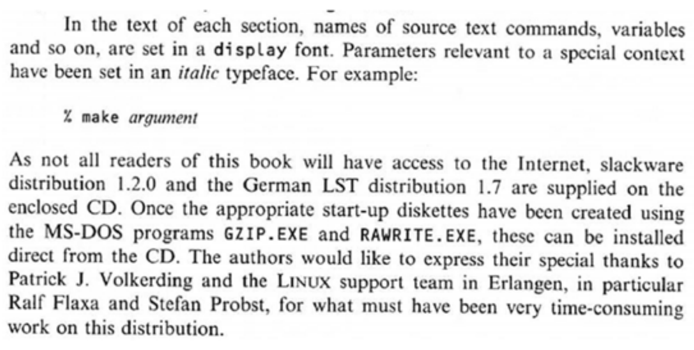
If you are installing Red Hat, then you will need to specify a root device and so on. So you would use:

```
MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
```

MILO will automatically contain the block devices that you configure into your vmlinux. I have tested the floppy driver, the IDE driver and a number of SCSI drivers (for example, the NCR 810), and these work fine. Also, it is important to set the host id of the SCSI controller to a reasonable value. By default, MILO will initialize it to the highest possible value (7) which should normally work just fine. However, if you wish, you can explicitly set the host id of the *n*-th SCSI controller in the system by setting environment variable `SCSIn_HOSTID` to the appropriate value. For example, to set the hostid of the first SCSI controller to 7, you can issue the following command at the MILO prompt:

```
setenv SCSI0_HOSTID 7
```

Linux Redhat, at Appendix E.6.2

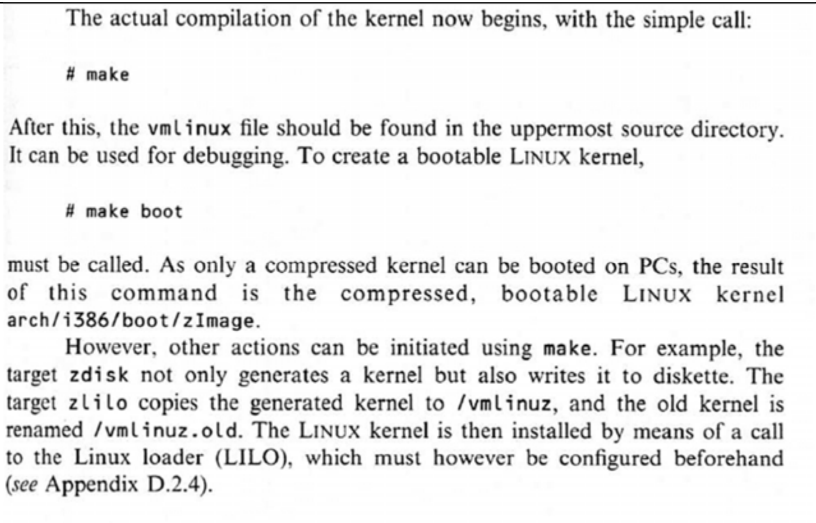


In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

`% make` *argument*

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface



The actual compilation of the kernel now begins, with the simple call:

```
# make
```

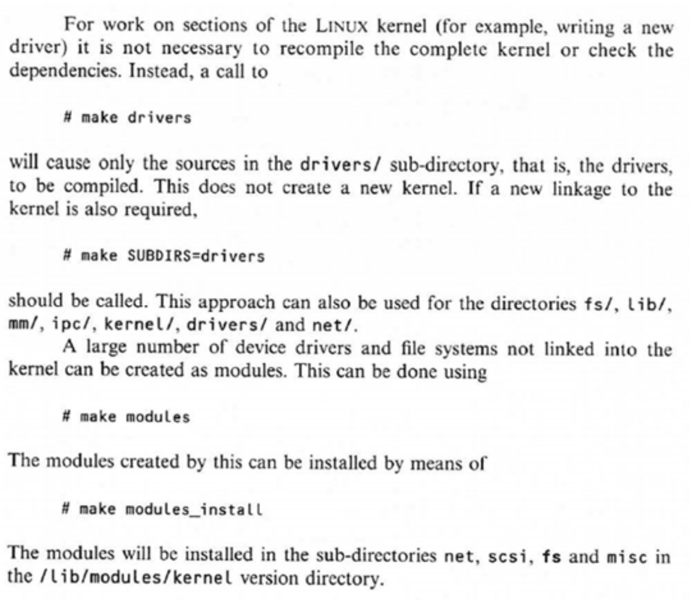
After this, the `vmlinux` file should be found in the uppermost source directory. It can be used for debugging. To create a bootable LINUX kernel,

```
# make boot
```

must be called. As only a compressed kernel can be booted on PCs, the result of this command is the compressed, bootable LINUX kernel `arch/i386/boot/zImage`.

However, other actions can be initiated using `make`. For example, the target `zdisk` not only generates a kernel but also writes it to diskette. The target `zlilo` copies the generated kernel to `/vmlinuz`, and the old kernel is renamed `/vmlinuz.old`. The LINUX kernel is then installed by means of a call to the Linux loader (LILO), which must however be configured beforehand (*see* Appendix D.2.4).

Beck, at 2.2



For work on sections of the LINUX kernel (for example, writing a new driver) it is not necessary to recompile the complete kernel or check the dependencies. Instead, a call to

```
# make drivers
```

will cause only the sources in the `drivers/` sub-directory, that is, the drivers, to be compiled. This does not create a new kernel. If a new linkage to the kernel is also required,

```
# make SUBDIRS=drivers
```

should be called. This approach can also be used for the directories `fs/`, `lib/`, `mm/`, `ipc/`, `kernel/`, `drivers/` and `net/`.

A large number of device drivers and file systems not linked into the kernel can be created as modules. This can be done using

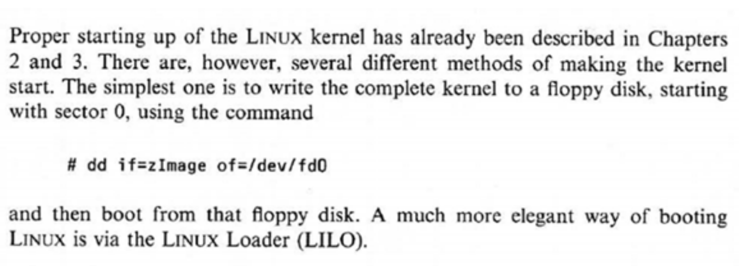
```
# make modules
```

The modules created by this can be installed by means of

```
# make modules_install
```

The modules will be installed in the sub-directories `net`, `scsi`, `fs` and `misc` in the `/lib/modules/kernel` version directory.

Beck, at 2.2

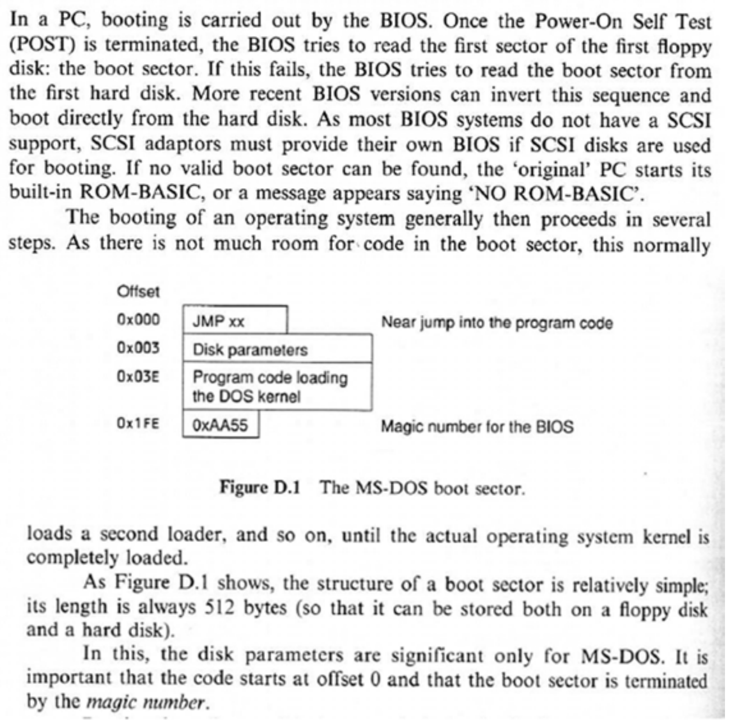


Proper starting up of the LINUX kernel has already been described in Chapters 2 and 3. There are, however, several different methods of making the kernel start. The simplest one is to write the complete kernel to a floppy disk, starting with sector 0, using the command

```
# dd if=zImage of=/dev/fd0
```

and then boot from that floppy disk. A much more elegant way of booting LINUX is via the LINUX Loader (LILO).

Beck, at Appendix D



In a PC, booting is carried out by the BIOS. Once the Power-On Self Test (POST) is terminated, the BIOS tries to read the first sector of the first floppy disk: the boot sector. If this fails, the BIOS tries to read the boot sector from the first hard disk. More recent BIOS versions can invert this sequence and boot directly from the hard disk. As most BIOS systems do not have a SCSI support, SCSI adaptors must provide their own BIOS if SCSI disks are used for booting. If no valid boot sector can be found, the 'original' PC starts its built-in ROM-BASIC, or a message appears saying 'NO ROM-BASIC'.

The booting of an operating system generally then proceeds in several steps. As there is not much room for code in the boot sector, this normally

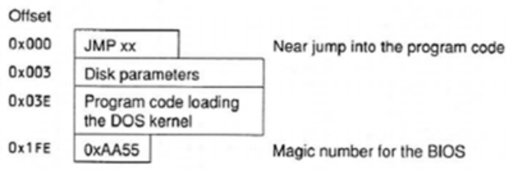

Offset
0x000
JMP xx
Near jump into the program code
0x003
Disk parameters
0x03E
Program code loading the DOS kernel
0x1FE
0xAA55
Magic number for the BIOS

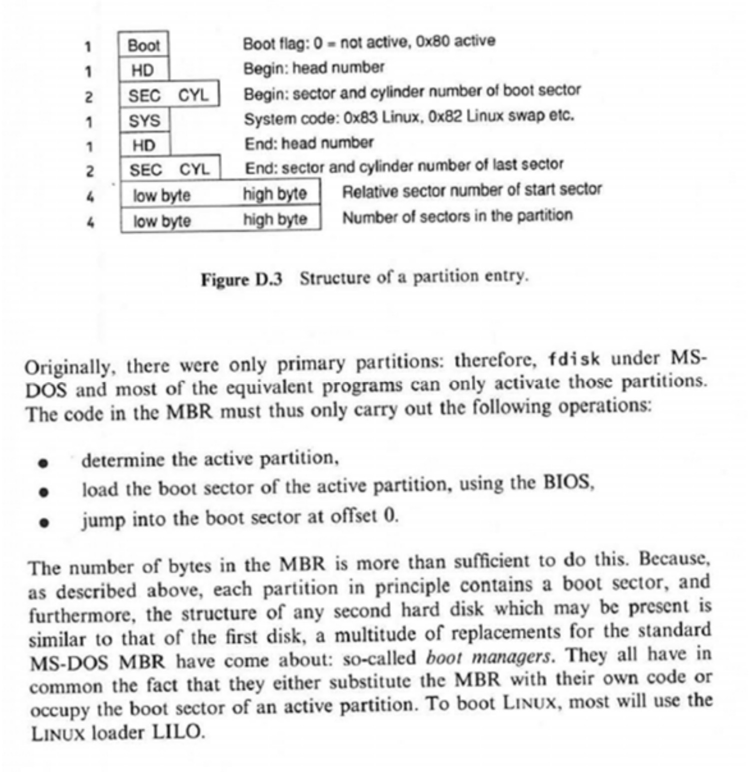
Figure D.1 The MS-DOS boot sector.

loads a second loader, and so on, until the actual operating system kernel is completely loaded.

As Figure D.1 shows, the structure of a boot sector is relatively simple; its length is always 512 bytes (so that it can be stored both on a floppy disk and a hard disk).

In this, the disk parameters are significant only for MS-DOS. It is important that the code starts at offset 0 and that the boot sector is terminated by the *magic number*.

Beck, at Appendix D.1



1	Boot	Boot flag: 0 = not active, 0x80 active
1	HD	Begin: head number
2	SEC CYL	Begin: sector and cylinder number of boot sector
1	SYS	System code: 0x83 Linux, 0x82 Linux swap etc.
1	HD	End: head number
2	SEC CYL	End: sector and cylinder number of last sector
4	low byte high byte	Relative sector number of start sector
4	low byte high byte	Number of sectors in the partition

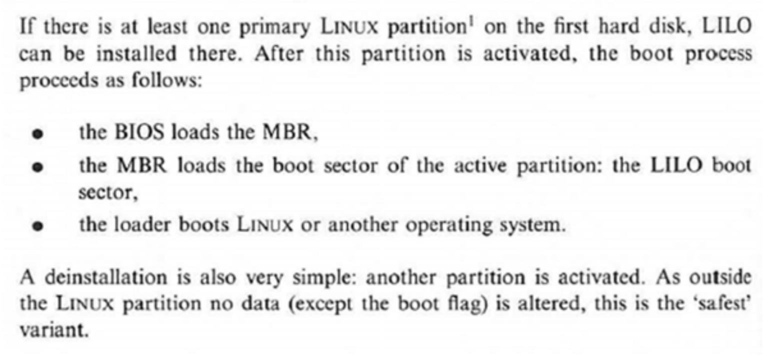
Figure D.3 Structure of a partition entry.

Originally, there were only primary partitions: therefore, `fdisk` under MS-DOS and most of the equivalent programs can only activate those partitions. The code in the MBR must thus only carry out the following operations:

- determine the active partition,
- load the boot sector of the active partition, using the BIOS,
- jump into the boot sector at offset 0.

The number of bytes in the MBR is more than sufficient to do this. Because, as described above, each partition in principle contains a boot sector, and furthermore, the structure of any second hard disk which may be present is similar to that of the first disk, a multitude of replacements for the standard MS-DOS MBR have come about: so-called *boot managers*. They all have in common the fact that they either substitute the MBR with their own code or occupy the boot sector of an active partition. To boot LINUX, most will use the LINUX loader LILO.

Beck, at Appendix D.1

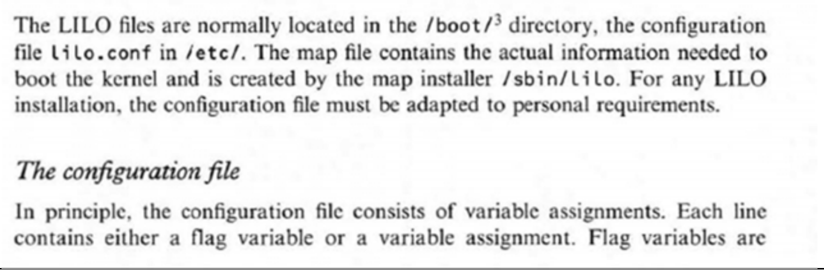


If there is at least one primary LINUX partition[1] on the first hard disk, LILO can be installed there. After this partition is activated, the boot process proceeds as follows:

- the BIOS loads the MBR,
- the MBR loads the boot sector of the active partition: the LILO boot sector,
- the loader boots LINUX or another operating system.

A deinstallation is also very simple: another partition is activated. As outside the LINUX partition no data (except the boot flag) is altered, this is the 'safest' variant.

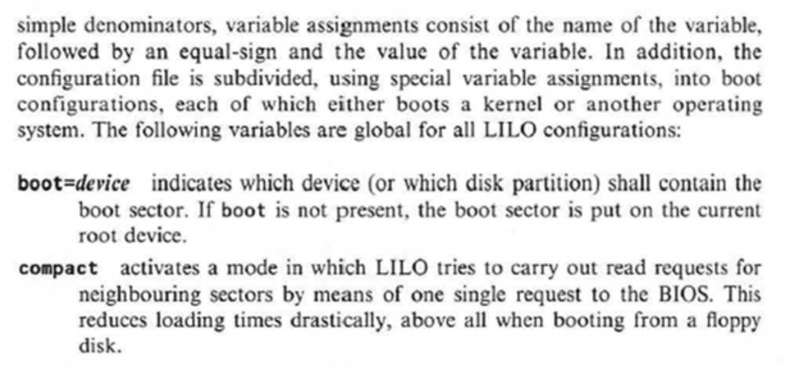
Beck, at Appendix D.2.1



The LILO files are normally located in the `/boot/`[3] directory, the configuration file `lilo.conf` in `/etc/`. The map file contains the actual information needed to boot the kernel and is created by the map installer `/sbin/lilo`. For any LILO installation, the configuration file must be adapted to personal requirements.

The configuration file

In principle, the configuration file consists of variable assignments. Each line contains either a flag variable or a variable assignment. Flag variables are



simple denominators, variable assignments consist of the name of the variable, followed by an equal-sign and the value of the variable. In addition, the configuration file is subdivided, using special variable assignments, into boot configurations, each of which either boots a kernel or another operating system. The following variables are global for all LILO configurations:

`boot=`*device* indicates which device (or which disk partition) shall contain the boot sector. If **`boot`** is not present, the boot sector is put on the current root device.

`compact` activates a mode in which LILO tries to carry out read requests for neighbouring sectors by means of one single request to the BIOS. This reduces loading times drastically, above all when booting from a floppy disk.

Beck, at Appendix D.2.4

17. The method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 15 above.	

19.1 The method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 14, wherein the request for the boot data comprises: a request to access boot data that is not associated with the boot data list, and wherein the updating comprises:), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1 and 1.2 above	

19.2 associating the accessed boot data that is not associated with the boot data list to the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "associating the accessed boot data that is not associated with the boot data list to the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, associating the accessed boot data that is not associated with the boot data list to the boot data list.), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 2 above.	

23. The method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 16 above.	

24. The method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system.
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the portion of the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 15 above.	

27. The method of claim 1, wherein the memory comprises: a physical memory.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein the memory comprises: a physical memory."

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

See Claims 1.1 and 1.2 above

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

28. The method of claim 1, wherein the operating system comprises: a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 16 above.	

29. The method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the boot data comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15 and 17 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

31. The method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein the accessing comprises: accessing the loaded portion of the boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1 and 1.2 above	

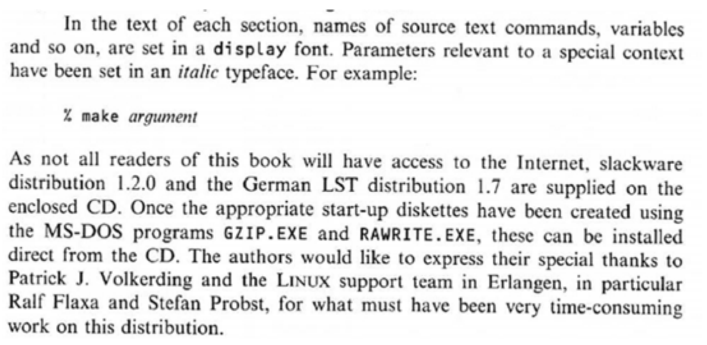
32. The method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein a form of dictionary encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

See Claim 1.1 above

See also


In the text of each section, names of source text commands, variables and so on, are set in a display font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs GZIP.EXE and RAWRITE.EXE, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

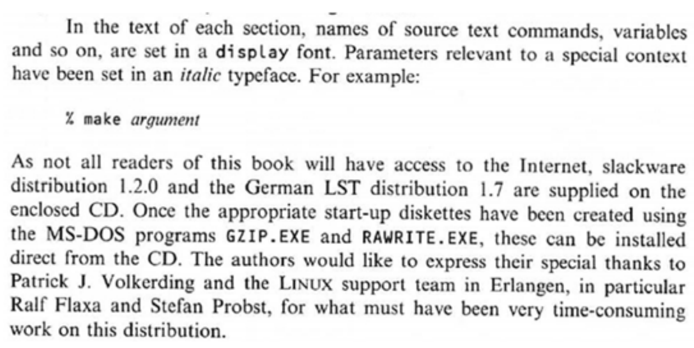
33. The method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form"

To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 1, wherein Lempel-Ziv encoding was utilized to encode the portion of the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions.

Linux Kernel discloses this limitation:

See Claims 1.1 and 32 above.

See also



In the text of each section, names of source text commands, variables and so on, are set in a `display` font. Parameters relevant to a special context have been set in an *italic* typeface. For example:

```
% make argument
```

As not all readers of this book will have access to the Internet, slackware distribution 1.2.0 and the German LST distribution 1.7 are supplied on the enclosed CD. Once the appropriate start-up diskettes have been created using the MS-DOS programs `GZIP.EXE` and `RAWRITE.EXE`, these can be installed direct from the CD. The authors would like to express their special thanks to Patrick J. Volkerding and the LINUX support team in Erlangen, in particular Ralf Flaxa and Stefan Probst, for what must have been very time-consuming work on this distribution.

Beck, at Preface

35. The method of claim 5, wherein the compressed boot data represents a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

36. The method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the compressed boot data comprises: a program code associated with an operating system of the computer system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15 and 17 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

39. The method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the loading comprises: loading the stored compressed boot data from the first memory to a second memory, and wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1 and 1.2 above	

40. The method of claim 36, wherein the operating system comprises: a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 36, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 36, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

43. The method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the accessing comprises: accessing the loaded compressed boot data via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 31 above.	

44. The method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 32 above.	

45. The method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 32 and 33 above.	

47. The system of claim 6, wherein the boot data in the compressed form represents a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

48. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system.	Linux Kernel, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

49. The system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form.	Linux Kernel, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: an encoder configured to compress the boot data to provide the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 10 above.	

50. The system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form.	Linux Kernel, as evidenced by the example citations below, discloses "the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, further comprising: a decoder configured to decompress the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 1 and 6 above.	

51. The system of claim 6, wherein the first memory comprises: a physical memory.	Linux Kernel, as evidenced by the example citations below, discloses "the system of claim 6, wherein the first memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the first memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 27 and 39 above.	

52. The system of claim 6, wherein the boot data the compressed form comprises: a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data the compressed form comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

53. The system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program.	Linux Kernel, as evidenced by the example citations below, discloses "the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the system of claim 6, wherein the boot data in the compressed form comprises: a program code associated with an operating system of the system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

59. The method of claim 8, wherein the operating system in the compressed form represents a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

60. The method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

63. The method of claim 8, wherein the second memory comprises: a physical memory.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 8, wherein the second memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the second memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 27 and 38 above.	

64. The method of claim 8, wherein the operating system comprises: a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

65. The method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the operating system in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

67. The method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 8, wherein the accessing comprises: accessing the loaded first portion from the second memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 31 above.	

71. The method of claim 11, wherein the boot data in the compressed form represents a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

72. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

75. The method of claim 11, wherein the memory comprises: a physical memory.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 27 above.	

76. The method of claim 11, wherein the operating system comprises: a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

77. The method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the boot data in the compressed form comprises: a program code associated with the operating system and an application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

79. The method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein the accessing comprises: accessing the boot data in the compressed form from the memory via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 31 above.	

80. The method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein a form of dictionary encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 32, 33 and 49 above.	

81. The method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 11, wherein Lempel-Ziv encoding was utilized to encode the boot data in the compressed form), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 32, 33 and 49 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

83. The method of claim 13, wherein the boot data in the compressed form represents a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form represents a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form represents a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

84. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

87. The method of claim 13, wherein the memory comprises: a physical memory.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein the memory comprises: a physical memory"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the memory comprises: a physical memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 27 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

88. The method of claim 13, wherein the operating system comprises: a plurality of files.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein the operating system comprises: a plurality of files"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the operating system comprises: a plurality of files), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 16 and 23 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

89. The method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the boot data in the compressed form comprises: a program code associated with the operating system and application program), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 15, 17 and 24 above.	

91. The method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein the accessing comprises: accessing the loaded boot data in the compressed form via direct memory access), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 31 above.	

92. The method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein a form of dictionary encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 32, 33 and 49 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

93. The method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 13, wherein Lempel-Ziv encoding was utilized to encode the compressed boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claim 32, 33, and 49 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

97.1 The method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list,	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, further comprising: accessing additional compressed boot data that is not associated with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 9.1-9.3 and 19 above.	

97.2 and wherein the updating comprises: associating the additional compressed boot data with the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "and wherein the updating comprises: associating the additional compressed boot data with the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, and wherein the updating comprises: associating the additional compressed boot data with the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.4.1, 2, 4, 5.6, and 8.6 above.	

98. The method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 5, wherein the updating comprises: disassociating non accessed boot data from the boot data list), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

107. The method of claim 2, further comprising: storing the updated boot list in a non-volatile memory.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory."
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: storing the updated boot list in a non-volatile memory), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.4.1, 5.6, and 8.6 above.	

108. The method of claim 2, further comprising: compressing at least a portion of the additional boot data.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 2, further comprising: compressing at least a portion of the additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 2, further comprising: compressing at least a portion of the additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 1.1, 5, 9.1 and 8 above.	

Appendix C38
Invalidity of U.S. Patent 8,880,862 based on Linux Kernel

109. The method of claim 108, further comprising: storing the compressed additional boot data.	Linux Kernel, as evidenced by the example citations below, discloses "the method of claim 108, further comprising: storing the compressed additional boot data"
To the extent that Realtime contends this reference does not explicitly disclose this element of this claim (for example, the method of claim 108, further comprising: storing the compressed additional boot data), Apple contends that one of skill in the art would understand the operation of booting a computer system to include the element that is missing similar to the manner in which the patentee relied upon such knowledge of skill in the art during prosecution. See Sections VI. and VII. of Apple's Invalidity Contentions. Linux Kernel discloses this limitation: *See* Claims 5.1, 8.1 and 9.1 above.	